TECHNICAL UNIVERSITY OF CRETE

DIPLOMA THESIS

Design and Implementation of a Low-Cost Portable Embedded Positioning System in Mixed Environments

Author:
Zisis CHAROKOPOS

Thesis Committee:
Prof. Apostolos DOLLAS
Prof. Michail G. LAGOUDAKIS
Prof. Panagiotis PARTSINEVELOS
(MRED, TUC)

A thesis submitted in fulfillment of the requirements for the diploma of Electrical and Computer Engineer

in the

School of Electrical and Computer Engineering
Microprocessor and Hardware Laboratory

Chania, May 2024

Πολυτεχνείο Κρήτης

Διπλωματική Εργασία

Σχεδίαση και Υλοποίηση Φορητού Ενσωματωμένου Συστήματος Προσδιορισμού Θέσης Χαμηλού Κόστους σε Μεικτά Περιβάλλοντα

Συγγραφέας:
Ζήσης ΧΑΡΟΚΟΠΟΣ

Εξεταστική Επιτροπή:
Καθ. Απόστολος ΔΟΛΛΑΣ
Καθ. Μιχαήλ Γ. ΛΑΓΟΥΔΑΚΗΣ
Καθ. Παναγιώτης ΠΑΡΤΣΙΝΕΒΕΛΟΣ
(ΜΗΧΟΠ, ΠΚ)

Διπλωματική εργασία που υποβλήθηκε για την εκπλήρωση των απαιτήσεων για το δίπλωμα του Ηλεκτρολόγου Μηχανικού και Μηχανικού Υπολογιστών

στη

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εργαστήριο Μικροεπεξεργαστών και Υλικού

Χανιά, Μάιος 2024

TECHNICAL UNIVERSITY OF CRETE

School of Electrical and Computer Engineering

Abstract

Design and Implementation of a Low-Cost Portable Embedded Positioning System in Mixed Environments

by Zisis CHAROKOPOS

In recent years, the development of positioning systems has been critical for many applications, including autonomous vehicles, robotics, and location-based services. Accurate positioning in mixed environments, as far as GNSS is concerned, is a fundamental requirement, but it can be challenging due to various factors, such as sensor limitations, real-time processing, and the complexity of the environment. Despite these challenges, this thesis proposes a positioning system that offers a promising solution. A mixed environment is considered an environment that fulfills the requirements for both GNSS-enabled and GNSS-denied environments. This system utilizes a stereo camera, an embedded system, and an accurate positioning algorithm and it achieves satisfactory results in specific scenarios. In this thesis, an initial approach of this system is implemented, and it achieves an accuracy, based on the distance difference, of 96.27% in high-featured small areas, and 90.44% in high-featured large areas, such features are used from the positioning algorithm.

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Περίληψη

Σχεδίαση και Υλοποίηση Φορητού Ενσωματωμένου Συστήματος Προσδιορισμού Θέσης Χαμηλού Κόστους σε Μεικτά Περιβάλλοντα

Ζήσης ΧΑΡΟΚΟΠΟΣ

Τα τελευταία χρόνια, η ανάπτυξη των συστημάτων εντοπισμού θέσης είναι σημαντική για πολλά πεδία εφαρμογών, συμπεριλαμβανομένων των αυτόνομων οχημάτων, της ρομποτικής και των υπηρεσιών που βασίζονται στην τοποθεσία. Ως μεικτό περιβάλλον θεωρείται ένα περιβάλλον που πληροί τις απαιτήσεις τόσο για περιβάλλοντα στα οποία μπορεί να χρησιμοποιηθεί δέκτης GNSS όσο και για περιβάλλοντα στα οποία αυτό δεν είναι δυνατό. Ο ακριβής εντοπισμός θέσης σε μεικτά, για το GNSS, περιβάλλοντα είναι θεμελιώδης λειτουργία, αλλά μπορεί να αποτελέσει πρόκληση λόγω διαφόρων παραγόντων, όπως οι περιορισμοί που προκύπτουν από τους αισθητήρες, η επεξεργασία σε πραγματικό χρόνο και η πολυπλοκότητα του περιβάλλοντος. Παρά τις προκλήσεις αυτές, το σύστημα εντοπισμού θέσης που προτείνεται στην παρούσα διπλωματική εργασία προσφέρει μια λύση που χρησιμοποιεί μια στερεοσκοπική κάμερα, ένα ενσωματωμένο σύστημα και έναν αλγόριθμο εντοπισμού θέσης με μεγάλη ακρίβεια που επιτυγχάνει ικανοποιητικά αποτελέσματα σε συγκεκριμένα περιβάλλοντα. Σε αυτή την διπλωματική εργασία, υλοποιείται μία αρχική προσέγγιση του συστήματος αυτού. Η ακρίβεια που επιτυγχάνεται, η οποία μετρήθηκε με βάση την διαφορά αποστάσεων, είναι 96,27% σε περιοχές που παρέχουν πολλά χαρακτηριστικά και είναι μικρής διάστασης. Καθώς επίσης, 90,44% σε περιοχές που παρέχουν πολλά χαρακτηριστικά και είναι μεγάλης διάστασης, τα οποία χαρακτηριστικά χρησιμεύουν στην διαδικασία προσδιορισμού θέσης από τον αλγόριθμο.

Acknowledgements

I want to express my sincere gratitude to all those who contributed to the successful completion of this thesis. First and foremost, I am thankful to my thesis supervisor, Prof. Apostolos Dollas, for his trust, and continuous support throughout the research and writing process. I would like to show my deepest appreciation to Prof. Panagiotis Partsinevelos, whose support and guidance played an important role in completing this thesis. Special thanks to Prof. Michail G. Lagoudakis for being a thesis committee member and assessing my work. I would like to especially thank the members of SenseLab, particularly Angelos Antonopoulos, for their immense support and guidance. I am deeply grateful to my family and friends for their persistent support and encouragement. At last, I would like to say a special thank you to my partner, Maria, for all her support and the patience she has shown. Also, for her assistance with grammar and spell-checking this work.

Contents

List of Figures

List of Tables

List of Abbreviations

3D	**3**-**D**imensional
5G	**5**th **G**eneration Mobile Networks
AoA	**A**ngle **o**f **A**rrival
AP	**A**ccess **P**oint
ASA	**A**crylic **S**tyrene **A**crylonitrile
BRIEF	**B**inary **R**obust **I**ndependent **E**lementary **F**eatures
CSV	**C**omma-**S**eparated **V**alues
CPU	**C**entral **P**rocessor **U**nit
DDR4	**D**ouble **D**ata **R**ate type **4** memory
FoV	**F**ield **o**f **V**iew
FTM	**F**ine **T**iming **M**easurement
GNSS	**G**lobal **N**avigation **S**atellite **S**ystem
GUI	**G**raphical **U**ser **I**nterface
HDMI	**H**igh-**D**efinition **M**ultimedia **I**nterface
IMU	**I**nertial **M**easurement **U**nit
IoT	**I**nternet **o**f **T**hings
IR	**I**nfra**R**ed
JS	**J**ava**S**cript
KNN	**K**-**N**earest **N**eighbor
LIDAR	**LI**ght **D**etection **A**nd **R**anging
LED	**L**ight **E**mitting **D**iod
LOS	**L**ine-**o**f-**S**ight
LPDDR4	**L**ow-**P**ower **D**ouble **D**ata **R**ate type **4** memory
MAE	**M**ean **A**bsolute **E**rror
ML	**M**achine **L**earning
MIPI CSI	**M**obile **I**ndustry **P**rocessor **I**nterface **C**amera **S**erial **I**nterface
MIPI DSI	**M**obile **I**ndustry **P**rocessor **I**nterface **D**isplay **S**erial **I**nterface
NLOS	**N**on-**L**ine-**o**f-**S**ight
OS	**O**perating **S**ystem
PLA	**P**oly **L**actic **A**cid
RF	**R**adio **F**requency

RAM	**R**andom **A**ccess **M**emory
RGB	**R**ed **G**reen **B**lue
RGBD	**R**ed **G**reen **B**lue **D**epth
ROS	**R**obotic **O**perating **S**ystem
RPi4	**R**aspberry **Pi** **4**th generation
RSSI	**R**eceived **S**ignal **S**trength **I**ndication
RTK	**R**eal **T**ime **K**inematic positioning
RTT	**R**ound-**T**rip **T**ime
SD	**S**ecure **D**igital
SDK	**S**oftware **D**evelopment **K**it
SDRAM	**S**ynchronous **D**ynamic **R**andom **A**ccess **M**emory
SIFT	**S**cale-**I**nvariant **F**eature **T**ransform
SLAM	**S**imultaneous **L**ocalization **A**nd **M**apping
SoC	**S**ystem-**o**n-**C**hip
SSD	**S**olid **S**tate **D**rive
SURF	**S**peeded-**U**p **R**obust **F**eatures
TDoA	**T**ime **D**ifference **o**f **A**rrival
TF	**T**rans**F**orm
ToA	**T**ime **o**f **A**rrival
UWB	**U**ltra-**W**ide**b**and
V-SLAM	**V**isual - **S**imultaneous **L**ocalization **A**nd **M**apping
VI-SLAM	**V**isual **I**nertial - **S**imultaneous **L**ocalization **A**nd **M**apping
VLC	**V**isible **L**ight **C**ommunication
VO	**V**isual **O**dometry

Chapter 1

Introduction

1.1 Motivation

In recent years, the development of positioning systems has been critical for many applications, including autonomous vehicles, robotics, and location-based services. Accurate indoor and outdoor positioning is a fundamental requirement, but it can be challenging due to various factors such as sensor limitations, real-time processing, and the complexity of the environment. Despite these challenges, the proposed positioning system discussed in this thesis offers a promising solution that utilizes a stereo camera, an embedded system, and an accurate positioning algorithm to achieve satisfactory results in specific scenarios. The performance metrics of this system are satisfying, particularly in high-featured small areas where the accuracy reaches 96.27%. In large high-featured areas, which naturally pose more challenges due to their scale and the increased potential for environmental interference, the system achieves 90.44% accuracy.

The objective was to develop a first-generation positioning device that would make use of a combination of multiple sensors. This device was intended to provide accurate positioning information in both indoor and outdoor environments. To accomplish this goal, a stereo camera was utilized as a primary sensor, and in parallel, the use of an Inertial Measurement Unit (IMU) was explored. The approach was based on the ORB-SLAM3 [1] algorithm for its high degree of accuracy and the support from the community. It is important to note that the accuracy levels reported by the authors of the ORB-SLAM3

algorithm are dependent on specific circumstances and datasets. These conditions may not hold true for different environments. Nevertheless, the potential of this algorithm was explored through its implementation on an embedded system and extensive testing in a variety of indoor and outdoor environments. The results of the testing were varied with accuracy levels varying on a case-by-case basis. These variations were largely due to differences in feature availability and distance in relation to the reference measurements. However, the proposed positioning device produced a level of accuracy that was promising for specific scenarios while, at the same time, leaving room for improvement.

1.2 Contributions of This Thesis

This thesis will discuss the design, implementation, and evaluation of the proposed positioning device. It will also provide recommendations for future improvements that can be made in order to further enhance the accuracy and reliability of the device. More specifically, the contributions of this thesis are:

- The development of a first-generation real-time system that is low-cost compared to the alternatives.
- The developed system performs accurate positioning and provides local coordinates.
- Performance evaluation of this system with promising results in small areas and satisfactory results in large areas.

1.3 Thesis Outline

The composition of the thesis is as follows:

- **Chapter 2 - Related Work:** Surveying tools and indoor positioning systems.
- **Chapter 3 - System Level Architecture:** Hardware components, software tools, and algorithms.
- **Chapter 4 - Design And Implementation Software:** Software components, graphical user interface.

- **Chapter 5 - Design And Implementation Hardware:** Raspberry Pi setup, component calibration, system structure through ROS.
- **Chapter 6 - System Integration, Verification, And Performance Evaluation:** System integration, IMU integration issue description, output rate improvement, performance evaluation through experiments.
- **Chapter 7 - Conclusions And Future Work**

Chapter 2

Related Work

An overview of the importance of topographic surveying and its applications in various industries such as construction, restoration, and mapping in discussed. Additionally, the historical background of surveying methods, including older techniques and ancient surveying instruments are presented. Modern surveying devices and methods such as the GNSS receiver, Total Geodetic Station, and 3D laser scanner are also presented, highlighting their principles of operation and accuracy. Furthermore, the limitations of these methods, particularly their suitability for indoor environments, and the specialized skills and costs associated with utilizing these tools are discussed.

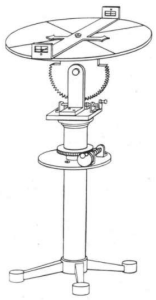
2.1 Topographic Surveying Methods

A scaled representation of a location, building, field, or area of interest, is called a topographic survey. Topographic surveys find usage in building construction, monument restoration, mapping, and other applications. The depiction produced by the survey must be as true as possible, resulting in the need for accurate methods and tools. Current and old topographic surveying methods will be described in this chapter as well as which one of them can be used for indoor environments.

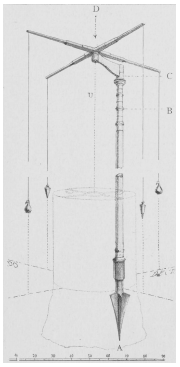
2.1.1 Older Topographic Surveying Devices and Methods

One of the oldest ways of surveying was a predecessor of the measuring tape, although before the standardization of the measuring units it didn't exist in the form we now know. The simplest form was a rope with knots in predefined distances. Using this tool was the simplest way of imprinting distances in scale. However, corners had to be represented too. To achieve

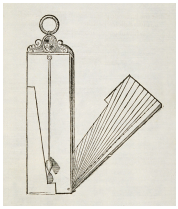
this, instruments like the dioptra [2], groma [3], and inclinometers [4] were utilized. They were the predecessors of the tools surveyors use today.



(A) Dioptra.[1]



(B) Groma.[2]



(C) Inclinometer.[3]

FIGURE 2.1: Older survey tools.

2.1.2 Modern Topographic Surveying Devices and Methods

In modern days, the need for more accurate depictions and the evolution of technology led to the discovery of the tools and methods used now. Some of the tools are the Total Geodetic Station, the 3-dimensional (3D) laser scanner, and the GNSS receiver. All these tools use the basic principle of acquiring points in the 3D space and connecting them with straight lines.

GNSS Receiver

The development of technology has led to the deployment of more and more satellites used for positioning. A tool that utilizes these satellites is the GNSS receiver. It receives signals from the satellites and calculates its position. A more accurate application is Real-time kinematic positioning (RTK), in which corrections are sent to the receiver for a specific area. It can achieve an accuracy of 1 to 3 cm. However, GNSS works only in environments where there is visual contact between the receiver and several satellites. The minimum number of satellites required for basic positioning is four, and as this number increases, the accuracy increases until it reaches 1 to 3 cm, as mentioned before. This portable and easy-to-use device is commonly used in surveying applications. However, it can only be used outdoors and is unsuitable for internal spaces.

[1]Source: https://en.wikipedia.org/wiki/Dioptra
[2]Source: https://en.wikipedia.org/wiki/Groma_(surveying)
[3]Source: https://en.wikipedia.org/wiki/Inclinometer

Total Geodetic Station

The total geodetic station is a device that utilizes an extremely accurate distance meter along with a high-precision angle measuring system. The system is based on the theodolite, which was first documented in 1571 [5]. Theodolite is a remarkably accurate angle and slope-measuring device. However, the total station is even more accurate, especially at longer distances, as it uses electronics to extract the information, minimizing human error. The total station is mounted on a tripod and must be leveled before use. The leveling procedure takes a few minutes and has to be repeated every time the total station is moved. It is very important and it is also time-consuming. The weight of the total station and its case is around 9 kg, whilst the tripod's weight varies from 4 kg to 10 kg, making transportation difficult, especially over uneven terrain, as seen in Figure 2.2. The total station can be used in combination with a prism, that is placed on top of a pole or another tripod. This prism is used to help acquire points that are not in the optical path, or as an easier way to acquire points, by setting the length of the pole in the calculations that the total station performs. However, using a prism requires the involvement of two people, one to operate the total station and the other to handle the prism. The total station provides the user with local (x,y,z) coordinates, which can later be referenced using a GNSS receiver. It is worth mentioning that this tool is primarily intended for outdoor use, although it can be used in large indoor areas. It is challenging, if not impossible, to use in narrow and confined spaces due to its size. The indoor alternative to the total station is measuring using a laser distance meter or a measuring tape. The usage of total stations requires specialization and the cost typically ranges between 2000€ and 12000€.

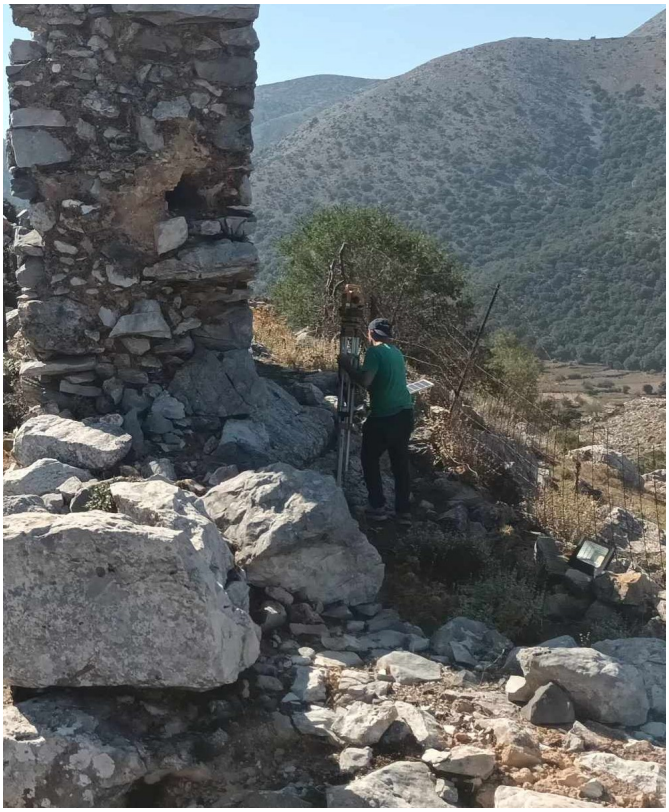


FIGURE 2.2: Surveyor carrying a Total Geodetic Station on uneven terrain during a monument restoration project.

3D Laser Scanner

An alternative to the traditional total station in surveying is the 3D laser scanner, which utilizes a system of cameras and a LIght Detection And Ranging (LIDAR) sensor to create a 3D model of the scanned area. The accuracy of the resulting 3D model is determined by the precision of the LIDAR sensor. In most 3D laser scanners, the points' density can be configured to save time in scanning and complexity in post-processing calculations. The density translates to how many points exist on a surface, which in turn means that the more points there are, the more accurate the model is. However, when there is a high number of points, the computational need is also higher. The user interface is usually an application from a tablet or smartphone. The process of creating the model is the following. Firstly the device is set on a tripod, which is set to be leveled. Secondly, the scanning can be started from the application, and a 3D model is created. These two steps are repeated until the whole desired area is covered. Between two consecutive stops, the models must have enough mutual points to connect. The final process is done

on a computer where the complete model is created by merging the individual models. In comparison to the total station, the 3D laser scanner is more accurate as it provides more points given that taking one measurement with the total station corresponds to one point. All that is needed for someone to be able to use it, is some basic training and familiarization with the application. The cost ranges from 5000€ to 90000€. In more known brands like Leica[6], the cost covers a spectrum between 18000€ and 90000€, making them extremely costly.

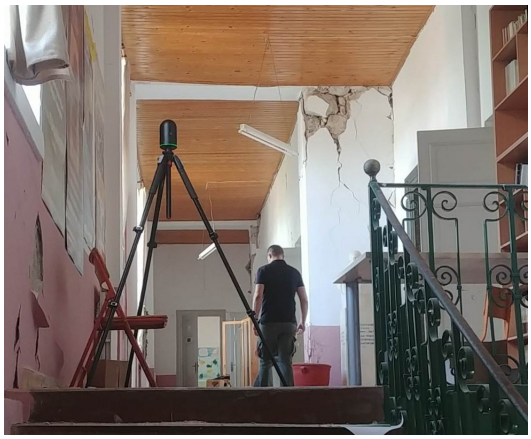


FIGURE 2.3: 3D Laser Scanner in a decommissioned school due to earthquake damage.

2.1.3 Surveying in GNSS-Denied Environments

It is important to categorize environments into four groups regarding their suitability for GNSS operation and size: large and small areas where GNSS can function accurately, large and small GNSS-denied environments. The first two categories include outdoor zones with an unobstructed view between the receiver and the satellites. GNSS-denied areas are places where the GNSS receiver cannot function due to the weak or absence of satellite signals. These areas can be either indoors or outdoors, such as rooms, corridors, gorges, areas under trees, and near high buildings. The GNSS receiver can be used in the first and second categories. It is preferred because of its ease of use. The total station is ideal for usage in the first, second, and third categories, and it is limited by its size for the fourth category. As far as the 3D laser scanner is concerned, it can be used in all the categories depending on the sensor, for example, BLK2GO[7] and BLK360[8] have a range of 25 m and 60 m respectively.

2.2 Indoor Positioning Systems

More specifically, in indoor positioning and navigation, several technologies and methods can be used which are discussed below.

2.2.1 Indoor Positioning Methods

The most common categories of methods are geometrical-based positioning, fingerprint-based positioning, and dead-reckoning.

Geometrical-Based Positioning

In the category of geometrical belongs the multi-lateration [9] process on which GNSS is based. This process attempts to compute the position of a device by finding the intersection of circles, or spheres in the 3D space, given the radius. Each sphere and radius corresponds to a base, with a known location, and the distance between the base and the device respectively, as seen in Figure 2.4. In order to accurately compute the position, it is imperative to have three or more bases. It is worth noting that the accuracy of the position increases proportionally with an increase in the number of bases used for computation. Also, the error of the distance between each base and the device affects the accuracy of the position. Higher distance error results in less accurate positioning.

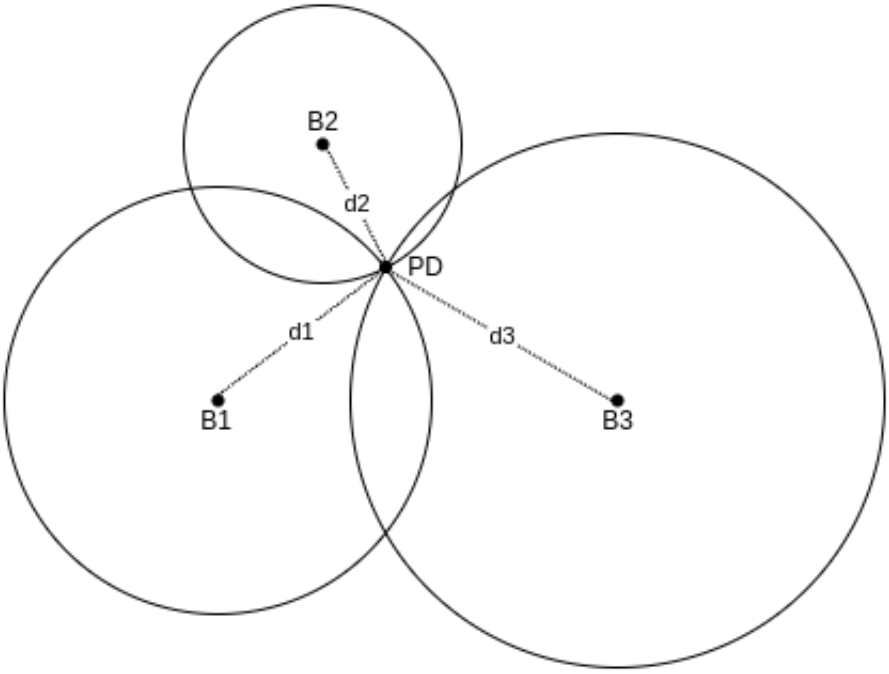


FIGURE 2.4: Example of lateration method. In this figure, there are a positioning device (PD), three bases (B1, B2, B3), and three distances between the PD and each base (d1, d2, d3). The intersection of the three circles is the location of the device.

Fingerprint-Based Positioning

Fingerprint-based positioning[10] has been considered as a technique that utilizes the distinctive features of certain parameters at specific points in 3D space. This technique consists of three phases, the first and the third are the offline and online phases respectively. During the offline phase, the system captures and associates the values of various parameters at different positions with the corresponding coordinates at those positions. This is done for each access point. Subsequently, the values obtained from all access points are combined for each position to create a map. During the online phase, the system receives a parameter value and matches it with a previously measured one, enabling it to estimate the location of the device. Additionally, the fingerprinting technique uses traditional machine learning and deep learning approaches to achieve better results [11].

Dead-Reckoning Positioning

Dead-reckoning is a method used in robotics, aviation, marine, and terrestrial navigation, among other fields to determine the position of a device by

calculating its movement from a known starting point. For the method to work, it is necessary to know the distance covered and the direction to estimate the current position. Despite its reliance on a straightforward principle, there are a few drawbacks associated with it. The most common one is the error accumulation. If an error occurs at early steps, this is propagated along the course and affects the estimation of the current position.

2.2.2 Parameters of Positioning Methods

It is important to understand the commonly used parameters before discussing technologies. These parameters include received signal strength (RSS), received signal strength indication (RSSI), round-trip time (RTT), time of arrival (ToA), time difference of arrival (TDoA), angle of arrival (AoA), and channel state information (CSI). RSS is a measurement of the signal strength that arrives at the receiver and is usually measured in dB or dBm. Meanwhile, RSSI is an indicator of RSS. RTT is a distance calculation method that relies on the time taken by a signal to travel from a device to an antenna and back. Two of the parameters belong to the time-of-flight (ToF) methods[10], namely ToA and TDoA. ToF methods exploit the time a wave takes to propagate and calculate distance. In the ToA method, the antenna sends a signal with a timestamp, and the receiver calculates the time difference between the received signal and the actual time. This difference is then multiplied by the speed of light, and the result is the distance. TDoA requires two perfectly synchronized stations because their time differences are used to extract the distance, as described in [10]. The AoA method utilizes directional antennas and calculates the angle according to the direction of the signal. The CSI [11] refers to the channel properties of a communication link. Other parameters can be extracted from it, such as AoA, ToA, or a combination of them. All of the aforementioned parameters can be used in geometrical-based as well as fingerprint-based positioning.

2.2.3 Radio Frequency Signals

The utilization of Radio Frequency (RF) signals for localization [12] has been commonly used. Signals such as Wi-Fi, Fifth Generation Mobile Networks (5G) [13], Bluetooth [14], and Ultra-Wideband (UWB) [15] are being exploited for both geometrical-based and fingerprinting-based positioning. These methods have proven to be effective in providing accurate and reliable location information.

More specifically, regarding the geometrical-based methods, the parameters that are used for each RF signal are shown in 2.1. Bluetooth can be remarkably power efficient, although the range can be smaller than other RF signals.

Signal	Parameters
Wi-Fi	RSS, RSSI, ToA, TDoA, AoA, CSI
5G	RTT, TDoA, AoA
Bluetooth	RSSI, TDoA, AoA
UWB	RSS, ToA, TDoA, AoA

TABLE 2.1: Parameters used for each technology.

Wi-Fi Fine Timing Measurement

A unique, among the RF geometrical-based methods, is Wi-Fi fine timing measurement (FTM) protocol. It was standardized by IEEE 802.11 [16] and it provides meter-level accuracy. FTM utilizes RTT as a ranging mechanism, which in line-of-sight (LOS) conditions is less affected by errors in comparison to RSSI. However, in non-line-of-sight (NLOS) conditions the accuracy is reduced and affected considerably by the environment. Another disadvantage is that FTM requires specific hardware. Currently, only a few smartphone models and Internet of Things (IoT) devices support FTM [12].

2.2.4 Acoustic Signals

Acoustic signals can provide an alternative positioning technology. This technology takes advantage of acoustic signals, commonly at the frequency band of 18-21 kHz, and it can utilize parameters such as RSS, ToA, and TDoA. The equipment that can be used is everyday speakers and microphones and can result in centimeter-level accuracy [17].

2.2.5 Pseudo-Satellite Positioning

Pseudo-satellite positioning technology uses ground-based pseudo-satellites to transmit signals for distance positioning, similar to those of a GNSS system. This makes pseudo-satellite technology a suitable candidate for smart cities [18]. The technology can be used for indoor positioning although it has some challenges, such as clock synchronization, multipath effects, and near-far interference. However, there is no need for different equipment than the one used for GNSS positioning. As in GNSS, pseudo-satellite positioning is

geometrical-based and is based on timing to provide localization. Various approaches and algorithms have been employed to achieve high-precision positioning in both static and dynamic scenarios [12].

2.2.6 Visible Light Communication Positioning

An extremely accurate technology is Visible Light Communication (VLC) positioning [19]. VLC initially used traditional methods such as k-nearest neighbor (KNN). However, because computational demands are high, nowadays VLC mainly takes advantage of machine learning (ML) algorithms. In order for these algorithms to be trained and tested, a dataset is required, and it can be created by utilizing RSS fingerprinting-based positioning. It is important to note that this technique achieves an accuracy of 0.8 cm [20].

2.2.7 Magnetic Field Positioning

The exploitation of the earth's magnetic field has led to the creation of magnetic field positioning [12]. The method used is fingerprinting-based, and the accuracy provided can be of meter level. Appart from the earth's magnetic field, artificial magnetic fields can be used by taking advantage of coils or magnets. According to [21], artificial magnetic fields can provide accuracy of up to millimeters in the range of meters.

2.2.8 Inertial Navigation

Through the use of an inertial measurement unit (IMU), a device's acceleration and orientation can be extracted, which can be highly beneficial for dead-reckoning positioning [12], as mentioned in 2.2.1, as it allows for distance and direction calculations. This technology, known as inertial navigation, has demonstrated significant utility. However, it is important to recognize that IMUs are inertial sensors that generate both random and constant errors, which will be explored further in 5.2. The primary obstacle is accurately forecasting and mitigating these errors, as they can have a critical impact on the dead-reckoning method.

2.2.9 Visual Odometry

The field of visual odometry (VO) relies on camera sensors to determine the location of a device, utilizing algorithms that address the simultaneous localization and mapping (SLAM) problem [12]. Through this method, VO estimates the position by cumulatively calculating the current location based on the prior one. However, like other dead-reckoning techniques, it suffers from the same limitations. VO finds applications mainly in the fields of robotics, ML, and autonomous driving among others.

2.3 Chapter Summary

It is important to note that for localization to occur, all RF methods, in addition to acoustic signals, pseudo-satellite, and VLC necessitate the existence of at least one base station installed. More specifically, Wi-Fi requires access points (APs), 5G relies on telecommunication antennas, acoustic signals take advantage of speakers, pseudo-satellites require ground stations, and VLC makes use of light sources such as light-emitting diodes (LEDs). The fact that they require base stations, makes these technologies unsuitable. Magnetic field positioning is not selected also for the reason that it requires mapping before trying to estimate the position, as it works using a fingerprinting-based method. In this thesis, it is essential that the system works in unknown environments. This results in excluding the technologies using the fingerprint-based methods, as well as the technologies requiring base stations. For this reason, VO has been chosen as positioning technology.

Chapter 3

System Level Architecture

In this chapter, the hardware devices and components as well as the algorithms and the software tools used, are presented and discussed.

3.1 Hardware Components

Before diving into the details of the system, a general overview would be helpful for understanding. The basic components of the system are a stereo camera and an embedded system, used for processing. For the user to control this system, a smartphone can be connected. The connection between the embedded and the stereo camera is achieved via USB, while the connection between the embedded and the smartphone is via WiFi as shown in Figure 3.1.

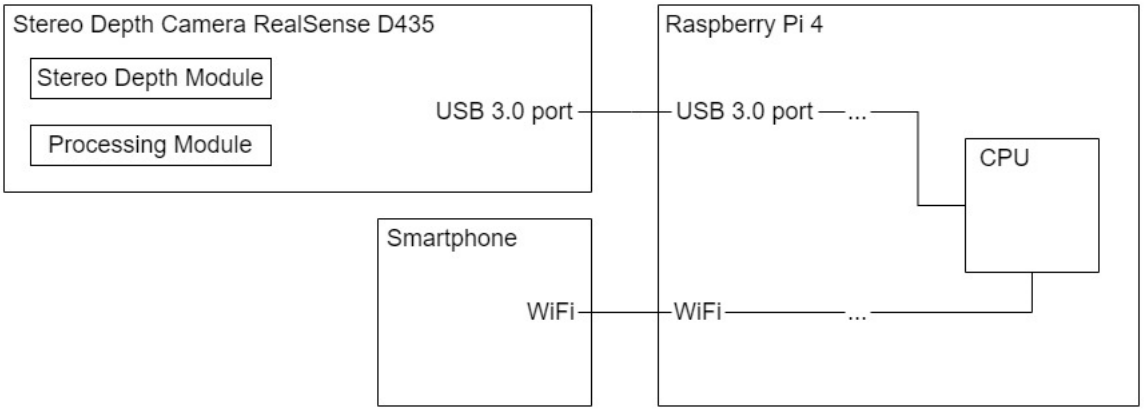


FIGURE 3.1: General block diagram of the final system.

It is important to note that the characteristics of two IMUs are discussed although they were not included in the final system. However, they were tested and the results are discussed in a later chapter.

3.1.1 Inertial Measurement Unit

An IMU is a sensor that provides data about its inertial status, which includes a variety of parameters such as accelerations, and angular velocities. The accelerometers measure the linear acceleration of the device, while the gyroscopes measure the angular rate and provide information on the rotation of the device. In addition, some IMUs also contain magnetometers which measure the strength and direction of the magnetic field to provide information about the device's orientation. The IMU is a key component in many applications such as navigation, aviation, control, robotics, and virtual reality, among others.

Pixhawk Cube Black Flight Controller

Pixhawk[22] is a flight controller that contains several sensors, including IMU, magnetometer, and barometer. In this thesis, the IMU is used to its full potential, with its data being accessed via a micro USB port. The data can be transmitted at a frequency of up to 330Hz. It is also worth mentioning that there is an abundance of input and output pins, some of which are USB, serial, SPI, I2C, CAN, CPPM, S. Bus, RSSI, DSM, and PWM among others. All the data from the aforementioned pins is being transferred from and to the FMU processor. Some of them are transferred through the I/O processor, as shown in Figure 3.2. In this thesis, the micro-USB is used, with its data being transferred directly to and from the FMU.

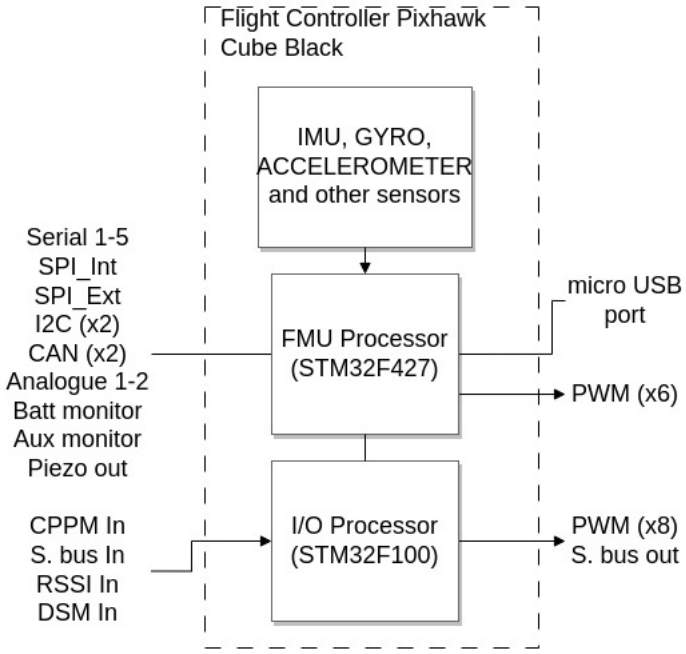


FIGURE 3.2: Pixhawk Cube black.

Intel® RealSense™ Depth Camera T265

Realsense T265[23] is a stereo camera that is equipped with an integrated IMU. The device features two cameras equipped with fisheye lenses, along with the IMU, as shown in Figure 3.3. ORB-SLAM3 crops the edges of the images taken by a large field of view (FoV) camera, as a measure of undistortion of the image. This cropping leads to the loss of valuable information and that is the reason this device is used only for its IMU. The accelerometer provides a frequency of 62.5Hz while the gyroscope provides 200Hz. The connection of this camera with the rest of the system was achieved with the use of the USB port.

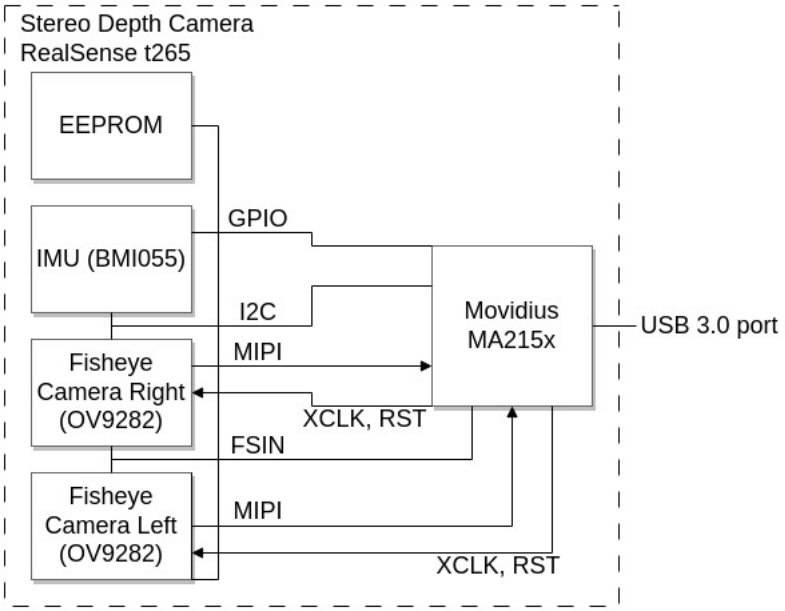


FIGURE 3.3: RealSense T265 stereo camera.

3.1.2 Visual System

Intel® RealSense™ Depth Camera D435

Realsense D435 [24] is a depth sensor that utilizes a stereo camera. The main components of this device are an IR projector, two cameras with no infrared (IR) filter, an RGB camera, and the Visual Processor D4 which is responsible for the controlling and management. It uses the projector to display dots which helps in feature detection, especially on plain surfaces, and then it calculates the depth based on the distance between these dots and the difference between the two images. In this thesis, it is used for its stereo camera attributes as well as for the depth stream. As far as the Vision Processor is concerned, it processes all the necessary data, including depth estimation, and transfers it via the USB port. The minimum supported distance varies for each resolution, in 848x480 is 195 mm, and the maximum distance can be more than 10 m, depending on the conditions. The supported resolution can be seen in table 3.1.

Format	Resolution	Frame Rate (fps)	Type of stream and lens
Y8 (8 bits)	1280x720	6, 15, 30	Left and Right Imager
Y8 (8 bits)	848x480, 640x480, 640x360, 480x270, 424x240	6, 15, 30, 60, 90	Left and Right Imager
YUV2 (16 bits)	1920x1080, 1280x720	6, 15, 30	Color Stream from RGB camera
YUV2 (16 bits)	848x480, 640x480, 640x360, 424x240	6, 15, 30, 60	Color Stream from RGB camera
YUV2 (16 bits)	320x240, 320x180	6, 30, 60	Color Stream from RGB camera
Z (16 bits)	1280x720	6, 15, 30	Depth
Z (16 bits)	848x480, 640x480, 640x360, 480x270, 424x240	6, 15, 30, 60, 90	Depth

TABLE 3.1: Supported resolutions and frame rates of RealSense D435. [24]

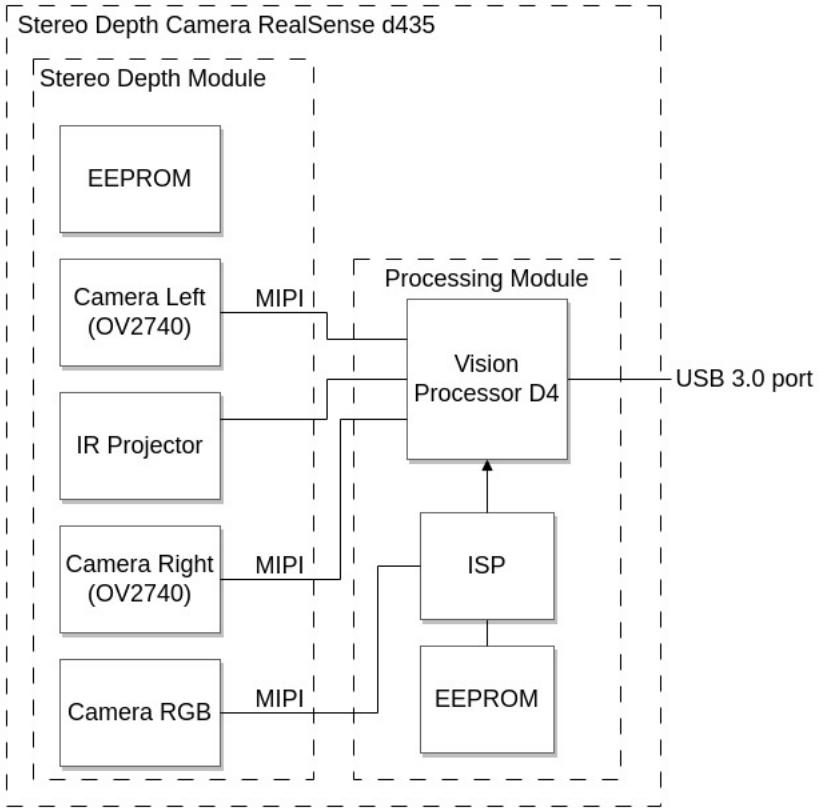


FIGURE 3.4: RealSense D435 stereo camera.

3.1.3 Development Platforms

The implementation was performed in two distinct phases. The first phase involved the initial development and testing of the ORB-SLAM3 algorithm, which was performed on a laptop, whose characteristics will be analyzed later. The second phase focused on integrating the tested algorithm into an embedded system, more specifically, the Raspberry Pi 4th generation (RPi4). Their hardware specifications are discussed below.

Intel® Core™ i7-9750H Processor

The experimentation was initially performed on a Dell Inspiron 15 7590 laptop[25] before being transferred to the RPi4. The laptop is equipped with a six-core, twelve-thread Intel® Core™ i7-9750H processor, 16 GB of RAM, and a 256 GB SSD. The aforementioned specifications indicate a quite powerful processor. The connection between the devices has been achieved using USB 3.0 and USB 2.0 interfaces, in detail the RealSense D435 and T265 require a

USB 3.0 connection, and the Pixhawk a USB 2.0. It is worth noting that this laptop is more powerful than the RPi4, however, it is also more expensive and significantly more power-consuming.

Raspberry Pi 4

The RPi4[26] is a low-cost and compact computer that has gained popularity in various fields, such as robotics and automation. One of its main advantages is that input and output ports are abundant, which makes it ideal for developing prototypes. However, only the USB ports were needed and used in this thesis. The block diagram in Figure 3.6 shows the RPi4's specifications[27]. The main module contains the processor and Video Processor 6 (VP6). The processor consists of a 4-core ARM Cortex-A72 64-bit System-on-Chip (SoC) that operates at 1.8GHz. However, the operating frequency was set at 2.2 GHz in order to utilize its full processing capacity. It is worth noting that overclocking a device can lead to a higher temperature, which was managed by installing a suitable heat sink and fan, as shown in Figure 3.5. The RPi4 version used in this thesis utilizes 8 GB of LPDDR4 SDRAM that operates at 3200MHz. As far as connectivity is concerned, there is the Raspberry Pi standard 40-pin GPIO header, two micro HDMI ports, a MIPI CSI camera port, a MIPI DSI display port, an audio jack, and the ones used the most are the Bluetooth and Wi-Fi module as well as the USB ports. The variety of all the connectivity makes this platform perfect for testing and prototyping. Specifically, two USB 2.0 and two USB 3.0 ports were used for this thesis. The data passes through the USB ports to the USB controller, which is connected to the CPU via PCIe, as shown in Figure 3.6.

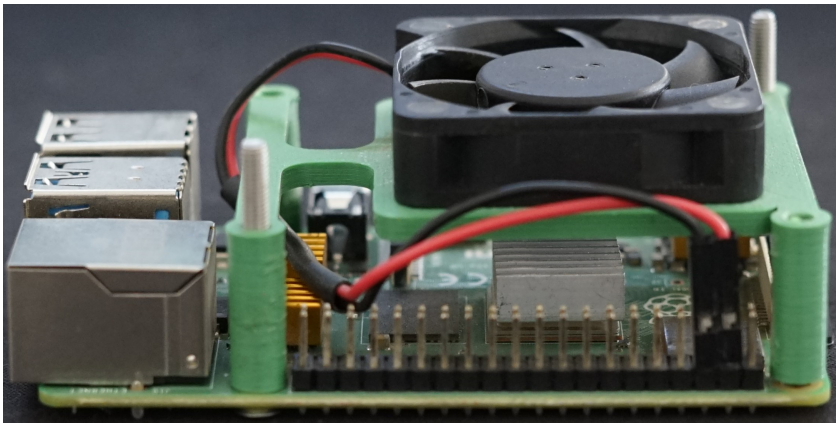


FIGURE 3.5: Raspberry Pi 4 with a heat sink and fan.

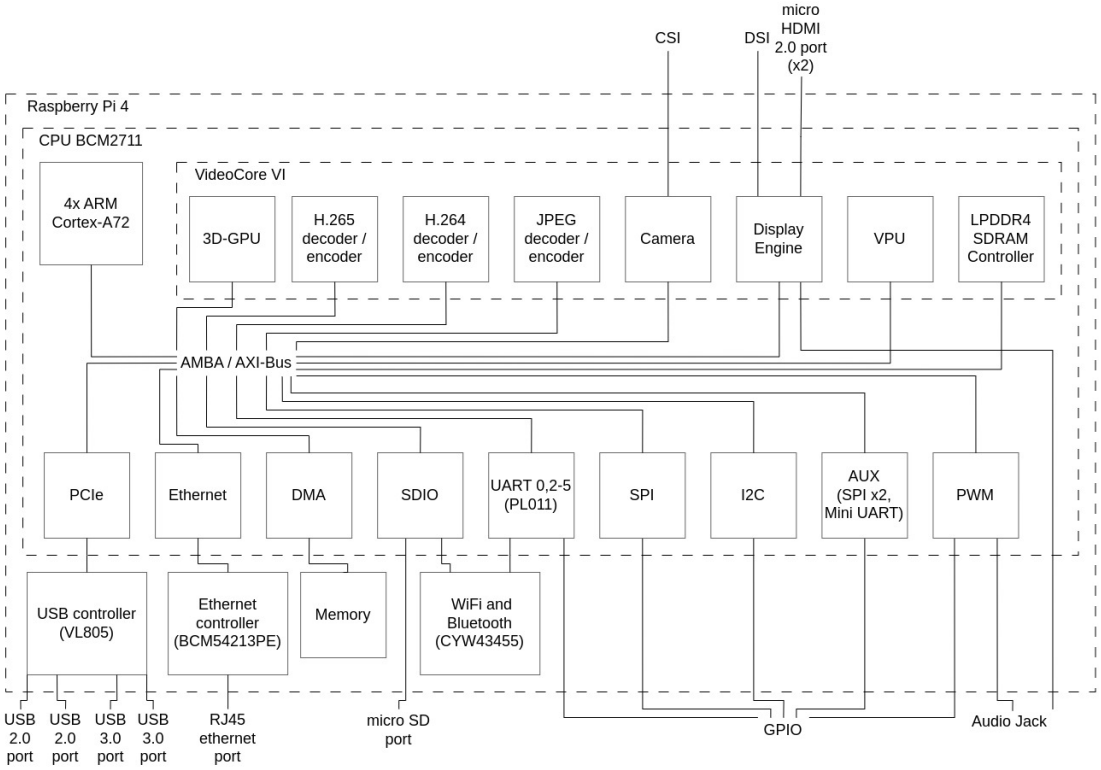


FIGURE 3.6: Raspberry Pi 4.

3.1.4 Final Setup

Here we can see how the devices are connected and the block diagram of the system as shown in Figure 3.7. In this block diagram, the unused components have been disregarded and only the used components are shown, from which the stereo camera is connected to the Raspberry Pi via USB and the smartphone via WiFi.

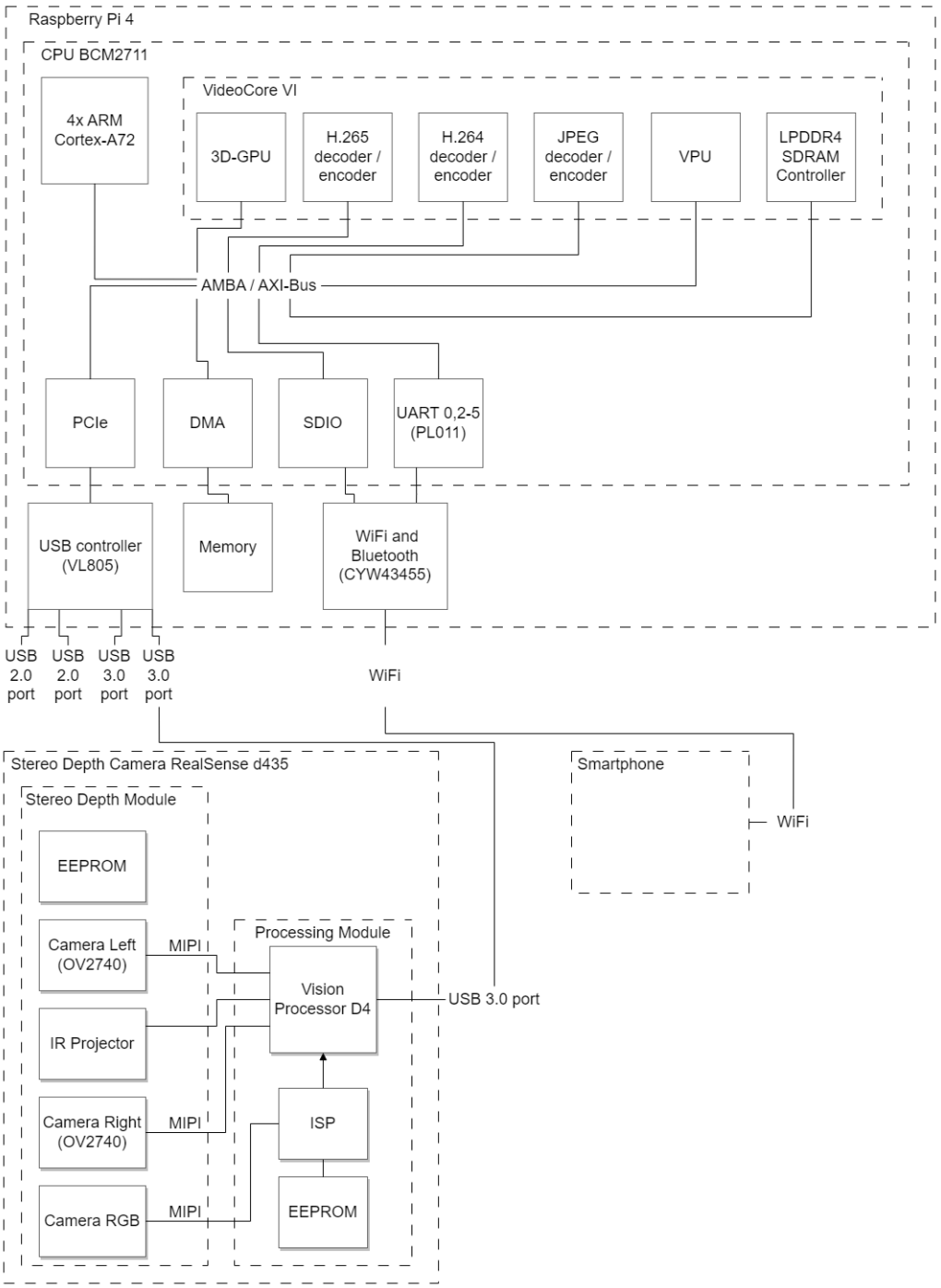


FIGURE 3.7: System block diagram.

3.2 Software Tools

3.2.1 OpenCV Library

The fields of robotics and computer science heavily rely on computer vision and machine learning. OpenCV is a powerful library that offers a wide range of tools for these areas, with support for C++ and Python programming languages. This thesis utilizes OpenCV through the use of the ORB-SLAM3 algorithm, which will be analyzed later.

3.2.2 ROS Framework

The Robotic Operating System (ROS)[28] is an open-source software framework that provides a set of libraries and tools essential for the effective management and communication of devices and sensors. It is particularly useful in developing autonomous systems that utilize one or more sensors. Communication between devices and sensors is accomplished through nodes, each of which can act as a publisher, a subscriber, or both. Publishers are responsible for sending data, while subscribers are nodes waiting to receive data. More specifically, nodes that acquire data from sensors are generally publishers, while nodes that manage sensors can be both publishers and subscribers. Messages are constructed using a standardized format. This can help to avoid every company having its own format of messages, leading to compatibility issues between devices. Messages are sent with an identifier that is unique for every type of message, called a topic. Each node can receive or send none, one, or many topics. The process of receiving data is referred to as *listening* to topics, while the process of sending data is called *publishing* on topics. Nodes can be written in either C or Python. -written nodes are compiled and executed using the *rosrun* command, while Python files are also executed using the same command. To better understand ROS, an example node is shown in Figure 3.8. In this example, the node *sensor_n* collects data from the sensor and publishes it using the topic */sensor/data*. The *data_processing_n* node subscribes to the */sensor/data* topic and publishes the */processed/data* topic, which the *data_display_n* node subscribes to. In this example, *sensor_n* is a publisher, *data_display_n* is a subscriber, and *data_processing_n* is both a publisher and a subscriber. This is a representation of a simple system, larger systems can have more complex structures.

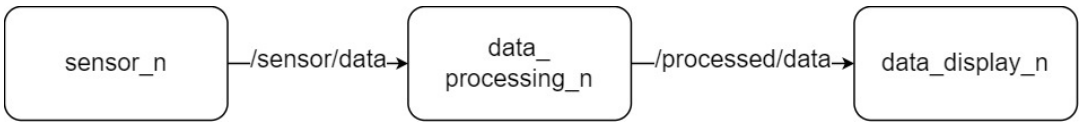


FIGURE 3.8: Example of ROS node usage.

It is worth noting that when a topic is published, it becomes visible to every node in the same system. To enable this communication, it is required for the *roscore* to be running. *Roscore* is a collection of nodes and programs that are responsible for initializing and maintaining communication. ROS provides packages that include the necessary nodes for a specific sensor installation. In this thesis, the following tools and commands have been utilized, *rosbag*, *catkin*, *rosbash*, *roslaunch*, and *rosservice*. *Catkin* is the build system of ROS. By running *catkin_make* all the programs defined by the CMakeFile.txt are being built. This command ensures that the programs, including the proper libraries, will be compiled. *Rosbag* is a command used to record a session of messages of topics in a file that has the *.bag* extension. The recorded file can be replayed, and the messages of the recorded topics are republished. It is an extremely useful tool since it allows the user to repeat an experiment without physically conducting it. *Rosbash* serves the same functionality as the *.bashrc* file in Ubuntu. It consists of a set of instructions that are executed each time a user logs in, which is crucial for initializing the system. *Rosrun* is a command used to execute programs and consequently, run nodes. By using *rosrun*, it is ensured that it will run in the ROS environment, although it requires *roscore* to already be running. *Roslaunch* is a command used to execute *.launch* files. These are files in the *xml* form used explicitly in ROS to organize and group other *launch* files or nodes. Additionally, they allow parsing arguments and parameters. These files do not require *roscore* to run, because they start a *roscore* if it is not running already. Lastly, a *rosservice* requires the existence of a server and a client to be performed. The server, which is contained in a node, waits for requests, or *calls* as named in ROS, while the client makes these *calls*. A *rosservice* can be called by a *launch* file, another node, or the user. It is often used for setting parameters while the node is running or by enabling or disabling streams.

Libraries Used

Another essential package for ROS is the transform (*TF*) library[29], which is provided by the package *tf2_ros*. This library is particularly useful for tracking the positions of components and establishing connections between them. In addition, the *CvBridge* package is a valuable package since it helps with the conversion between the ROS Image messages and OpenCV images. This conversion is necessary to enable the use of OpenCV's powerful tools for image processing. Another essential tool in the ROS ecosystem is *rviz*, which provides a 3D representation and orientation of devices or their components. It can be connected with the *TF* library to display the transformations and their connections. Together, these tools form a powerful suite of capabilities for building complex robotic systems using ROS.

3.3 Depth Estimation and SLAM

In this section, the depth estimation process and the algorithm for SLAM are described.

3.3.1 Depth Estimation

Depth estimation is a crucial task in computer vision and can be achieved using various sensors. An ultrasonic sensor is considered to be one of the cheapest options but has a limited detection range and low resolution. On the other hand, the LIDAR and the stereo camera are two sensors that provide a higher detection range and are more accurate. From them, a point cloud can be produced. In this thesis, the focus will be on the stereo camera for depth estimation. To obtain depth information, a stereo camera captures one photo from each camera. Feature detection and extraction are then performed on each image. During this process, specific features such as corners, edges, lines, or blobs are extracted depending on the algorithm used. The obtained features are then matched between the two photos and the depth information is estimated.

Feature Extraction and Matching

Various feature extraction methods have been developed for image processing, including Scale-Invariant Feature Transform (SIFT)[30] and Speeded-Up

Robust Features (SURF)[31]. SIFT is a scale-invariant algorithm that is capable of extracting features from an image, such as corners, for example, even if we zoom in on the image so much that the corner becomes a curve. While SURF was created as a faster solution to SIFT. FAST Algorithm for Corner Detection[32] fulfills the need of real-time applications, as it is faster than the previous. SIFT uses a lot of memory and can be time-consuming for matching features. This memory and time consumption is reduced by Binary Robust Independent Elementary Features (BRIEF)[33] and with a more increased recognition rate. Another notable method is the combination of FAST and BRIEF, known as Oriented FAST and Rotated BRIEF (ORB)[34], which was designed as a cost-effective alternative to SIFT and SURF, which are patented and not free to use. ORB is used by the ORB-SLAM3 algorithm, which is utilized for this thesis implementation. It is important to note that feature detection algorithms are unable to detect any features in a plain image as there are none to detect.

Disparity Map

Feature detection and extraction are applied in-depth estimation for the purpose of having unique areas of pixels in the image. This is helpful because the same features have to be found in both left and right images of the stereo camera. Then, the positions of these features in the left and right pictures are compared, and the horizontal difference in pixels is calculated. Since the cameras are already vertically aligned, there is no need to calculate the vertical difference, resulting in this difference being zero. The distance of each feature from the sensor is determined based on the number of pixels they are separated by, where a greater difference corresponds to a shorter distance. The disparity map is a new image in which each pixel represents the intensity of the difference in pixels mentioned before. With the use of the disparity map, and some of the characteristics of the camera, the distance from the camera is calculated. As shown in Figure 3.9, the Field of View (FoV) and the distance between the two sensors (Baseline) of the stereo camera, play a crucial role in the calculations.

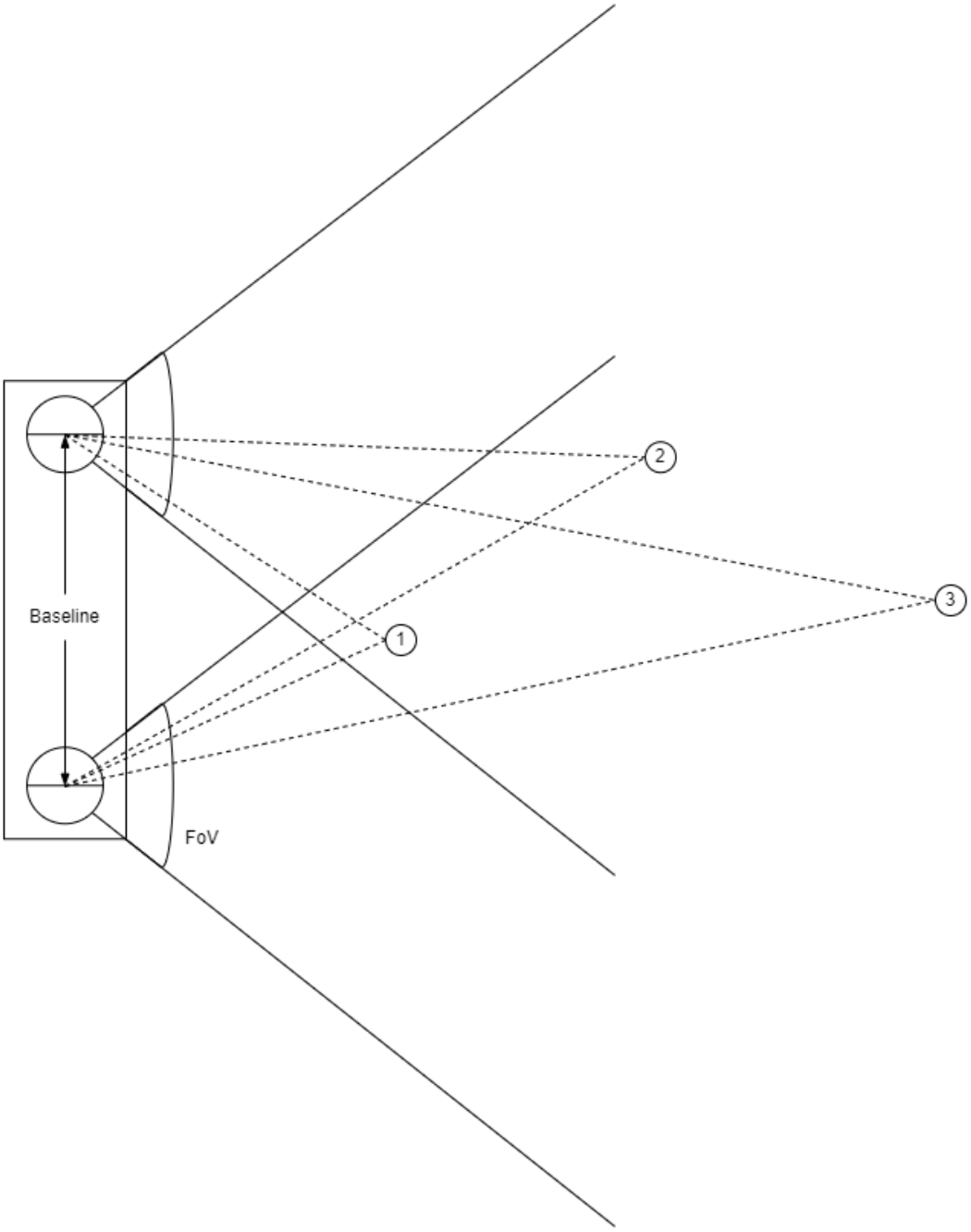


FIGURE 3.9: Distance of objects and baseline.

3.3.2 SLAM

The utilization of algorithms for SLAM [35] in robotics and computer vision is essential. This is happening because of the necessity of localization in GNSS-denied environments. An algorithm for SLAM generates a map of the environment and accurately determines the location of the device within the

generated map. Mapping can be carried out using sensors like ultrasonic, LIDAR, cameras, or any other ranging sensor, even combinations of them[36]. LIDAR provides the most accurate data while being the most expensive of them. In this thesis, a camera is chosen for its higher precision compared to an ultrasonic sensor and its lower cost compared to a LIDAR. Furthermore, unlike LIDAR, which is an active sensor that emits rays to calculate distance, a camera, being a passive sensor, does not require such capability. As the price of a LIDAR sensor is relatively high, passive sensors, particularly cameras, are more cost-effective to implement in algorithms for SLAM. Some of these algorithms utilize visual sensors as input to generate a map of the environment, these are called Visual-SLAM (V-SLAM). Correspondingly, when additional sensors such as IMUs are added, they are called Visual Inertial-SLAM (VI-SLAM). VI-SLAM algorithms enable the merging of data from inertial sensors with camera data.

3.3.3 ORB-SLAM3

Among the various V-SLAM approaches, ORB-SLAM[37] is a feature-based algorithm that utilizes feature detection and extraction methods integrated from the OpenCV library. As the name suggests, ORB-SLAM, and ORB-SLAM2[38] make use of ORB features. The latest version, ORB-SLAM3[1], also utilizes these features. ORB-SLAM3 is a state-of-the-art VI-SLAM algorithm that utilizes computer vision techniques to track the position and orientation of a camera in real-time. ORB-SLAM3 provides various configurations which are categorized by the devices used. The Mono and Stereo configurations utilize a single and a stereo camera respectively. The same two configurations can be combined with an IMU resulting in the configurations Mono-Inertial and Stereo-Inertial. Lastly, there is the Red Green Blue Depth (RGBD) configuration which utilizes an RGB camera with a depth stream. This algorithm was chosen for its accuracy as it can provide up to 9 millimeters accuracy, depending on the conditions[1]. Additionally, it is one of the most trustworthy algorithms [39], among others such as VINS-Mono [40].

3.4 Discussion

Another configuration among the components could include an IMU device. This was considered in the context of this thesis; however, due to it being a first-generation system, it was not utilized for the reasons described in 6.2.

Chapter 4

Design And Implementation Software

4.1 Software Components

The fundamental software components contained in the system include the operating system and the ROS, which oversees the majority of them. The operating system is responsible for managing the network, WiFi connection, and consequently the Graphical User Interface (GUI) server. Meanwhile, ROS is tasked with handling communication with the stereo camera, integrating the ORB-SLAM3 algorithm, connecting the GUI with the operating system, establishing links with custom-created nodes and other utilized packages, and managing overall internal communication, as shown in Figure 4.1.

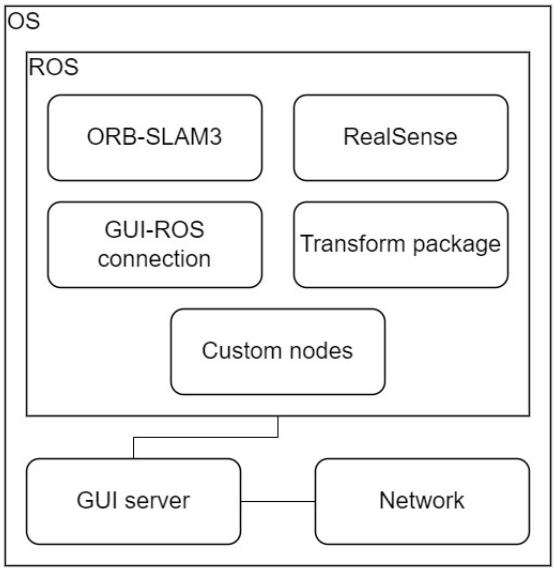


FIGURE 4.1: Software components.

4.1.1 Understanding of Depth Estimation

Before addressing the details of the system, and in order to gain a better understanding of depth estimation and feature detection and extraction, several experiments were conducted using functions from MATLAB. A representation of the depth estimation procedure follows. An image was taken from each sensor of the RealSense D435 stereo camera. These images are shown in Figure 4.2.

(A) Left image, before disparity map. (B) Right image, before disparity map.

FIGURE 4.2: Left and right images used for disparity map.

Then, feature extraction was performed, followed by feature matching. Feature matching is the procedure where the features from the left image are compared with the ones from the right image and their horizontal difference is calculated as shown in Figure 4.3.

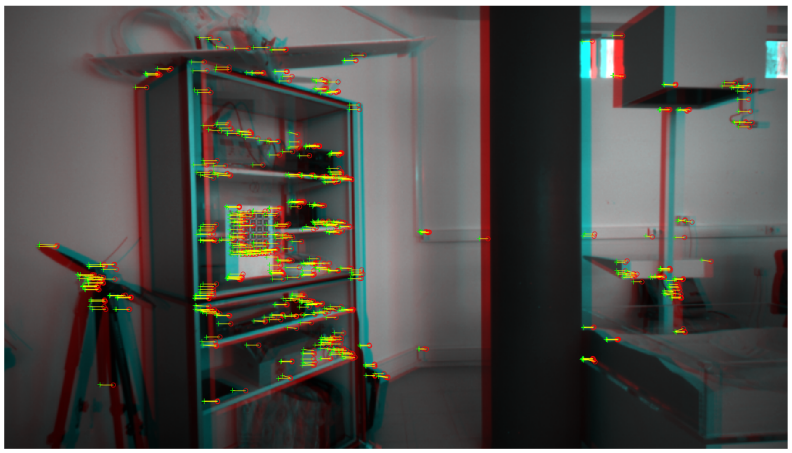


FIGURE 4.3: Difference in features.

After that, the value of these differences is calculated and the disparity map is created as shown in Figure 4.4.

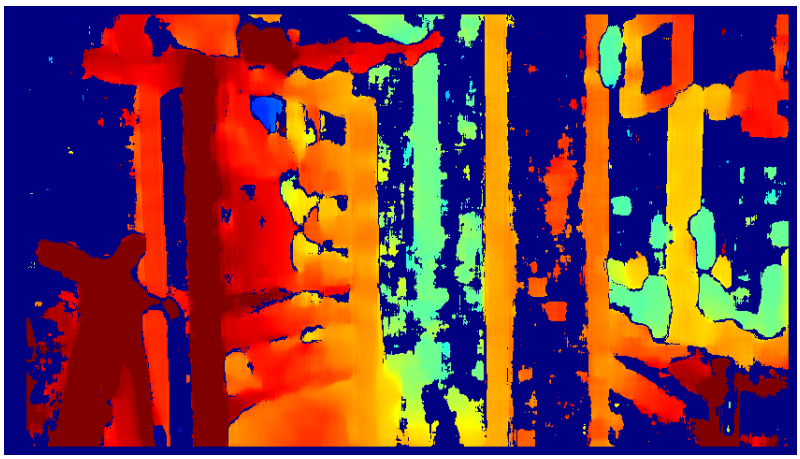


FIGURE 4.4: Disparity map.

To smooth the image, a filter based on Weighted Least Squares has been applied to the disparity map as shown in Figure 4.5. The nearest objects are depicted with red color and the farthest with blue.

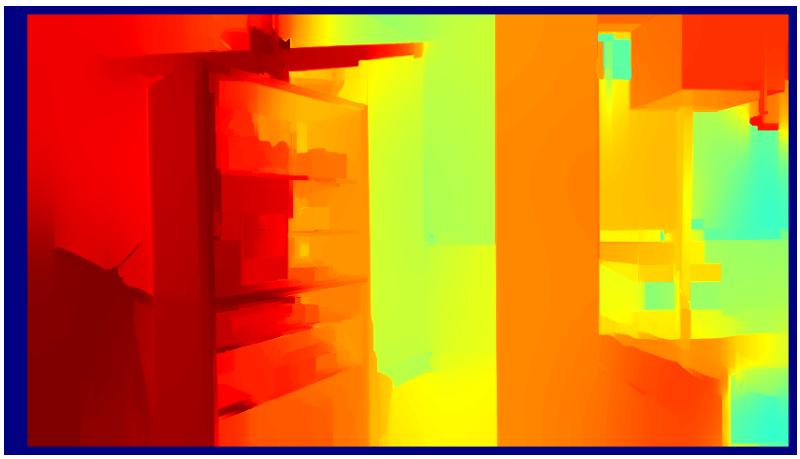


FIGURE 4.5: Filtered disparity map.

4.1.2 Graphical User Interface

In addition to the previous implementations, a basic GUI has been developed to support the system's functionality. The GUI is designed to assist user interaction with the system and display the captured points.

Computer GUI Implementation

It was initially created on a laptop before being redeveloped for the RPi4. It has been implemented using the wxPython library[41] and has been written in Python. The GUI interface has a list that displays the captured points, as shown in Figure 4.6. Below the list, users can select the type of point. This function enables users to switch between different point types while passing through a specific area only once. The GUI is equipped with several buttons with different functionalities, such as adding a new point, deleting the last entry, creating a new log file, and closing the program. The *Acquire Point*, *Delete Last Entry*, and *Close* buttons functions are self-explanatory. The *New Log File* provides the user with the ability to create a separate file, allowing them to organize the files as required.

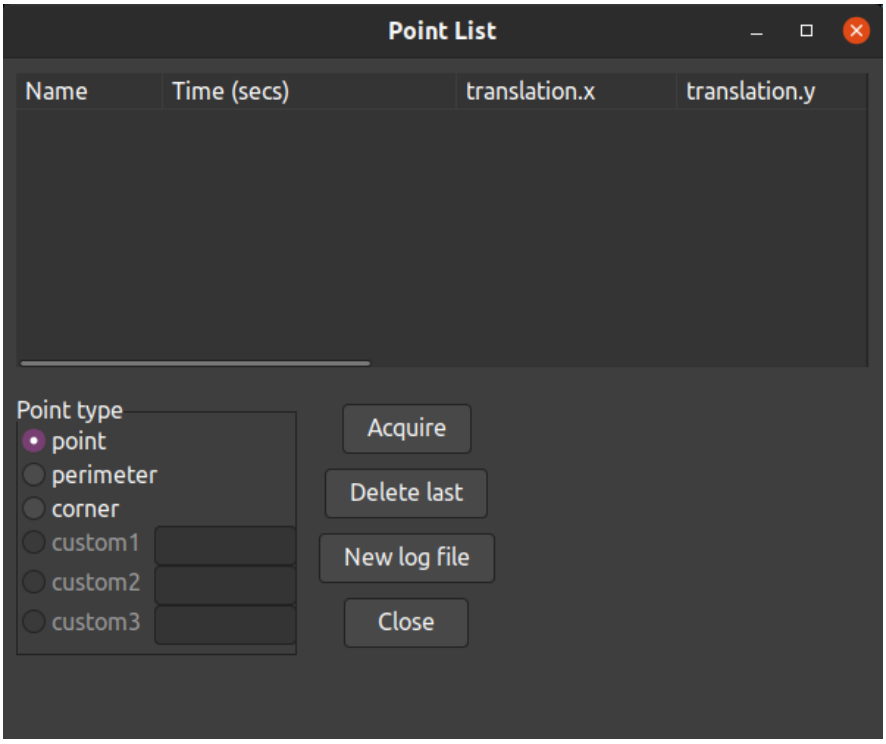


FIGURE 4.6: Graphical User Interface using wxPython.

Raspberry Pi GUI Implementation

The second version was developed in the form of a website. This form was chosen as it is more convenient to operate from a wide range of devices, including smartphones and laptops, without requiring any installation. The smartphone is connected remotely via WiFi with the RPi4, which operates

as a server. An HTML page has been created, with the system functionalities written in JavaScript (JS). These functionalities contain point capturing, deletion of the last captured point, deletion of all the captured points, and downloading a file of the captured points in comma-separated values (CSV) format. Another useful feature is the ability to rename the points. The user can select a name from a drop-down menu, which can be customized based on their requirements. his is useful because they might want to switch between *perimeter points* to *corner points* or any custom point. Another functionality that has been added to the previous version, is that the user can choose the length of the pole by inserting the value in the proper field. However, it is important to note that although a negative value can be assigned, it will be converted to a positive value. This is done for the reason that a convention had to be made. One option was to accept negative values of the length of the pole, in case a user would like to use it upside down. This could be done to capture points in a ceiling. Such a configuration could be achieved by placing the system between two poles. When the user wants to capture points on the ground, they would set the length with a positive value, otherwise, they would set it with a negative value. The other option was to ignore negative values and keep the coordinates as it is. In the case the user wants to measure points at the ceiling, all they have to do is turn it upside down. The second one was chosen because of its simplicity. Lastly, there is a checkbox that enables the algorithm to perform only localization. A snapshot of the interface is shown in Figure 4.7.

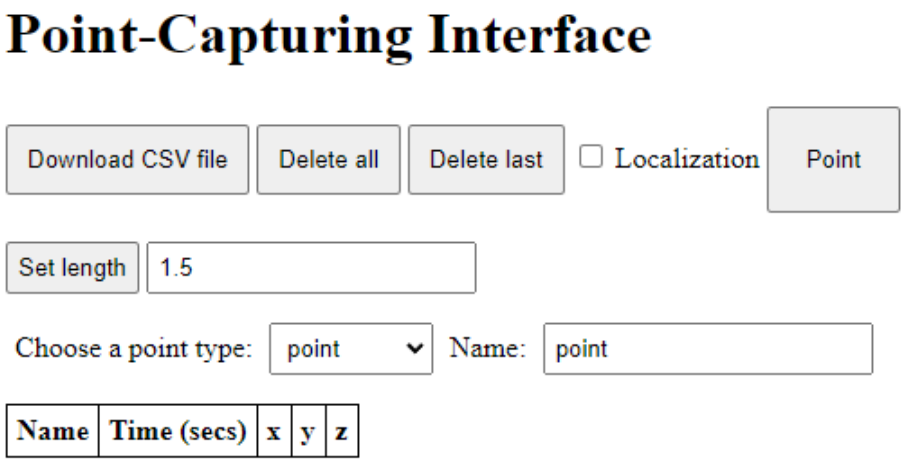


FIGURE 4.7: Final Graphical User Interface.

A more detailed description of the GUI implementation follows. A link between the JS program and the ROS needs to be established, and it is done using a package called *rosbridge_server*. This package provides several capabilities such as creating topics, subscribing to topics and publishing, and executing *rosservice* calls. For this particular implementation, the functionalities of subscribing to a topic and calling a service were utilized. The JS program creates a subscriber that listens to the topic that publishes the coordinates of the end of the pole. Moreover, the functionalities of the buttons have been implemented such that when the user clicks on the *Point* button, the program subscribes to the topic, and the message of the topic is loaded into a string variable. The user can select the point type from the drop-down menu, which includes three predefined names *point*, *perimeter*, and *corner*, as well as a custom one. If the user selects the *custom* option, they can enter a custom name for the point in the text field labeled *Name*. When the checkbox labeled "Localization" is checked, a *rosservice* call is executed, which causes the algorithm to stop the mapping functionality and perform only localization. This feature is important because when the algorithm performs only localization the computational needs are reduced. However, it is important to enable this option only after a loop closure has occurred and not before. Furthermore, the user can set the length of the pole by entering the desired value in the text field. When the user clicks on the *Set Length* button, a *rosservice* is called, which sends the value to the *nail_publisher* node. If the user clicks on the *Delete last* button, the latest point is deleted, and if they click on the *Delete all* button, all the points are deleted. Finally, when the user clicks on the *Download CSV file* button, a CSV file is created using the data from the captured points, and the file is sent to the user's device for download. This feature is especially practical because the user can later send the data to a computer. Notably, the GUI is accessible from any device that can connect to the WiFi or Ethernet port of the RPi4, including laptops or smartphones.

Chapter 5

Design And Implementation Hardware

The crucial steps involved in setting up the Raspberry Pi 4, including the installation of the operating system, remote access configuration, overclocking, and cooling are discussed. Additionally, the installation of ROS, RealSense SDK integration, and the installation of ORB-SLAM3 alongside its requirements are covered. This chapter provides a comprehensive overview of the hardware design and implementation process. It is worth noting that the implementation for the IMU, which is described here, was not included in the final configuration.

5.1 Raspberry Pi 4 Setup

5.1.1 Raspberry Pi Configuration and Setup

For this implementation, an RPi4 is being utilized as the final development platform. The previous tools have also been installed on the RPi4. However, the initial setup of the RPi4 is a crucial process that requires careful attention to prevent any potential errors. Following is an overview of the configuration that was selected and the actions that were taken.

Operating System Setup

The RPi4 uses the micro Secure Digital (SD) memory card as its primary storage device, where both the operating system (OS) and data are stored. To load an OS in the SD card, it needs to be flashed using various parameters. For this instance, Ubuntu Server 20.04 has been selected as the OS. The user

interface is conducted through the terminal, which eliminates the need for graphics, reducing the computational demands caused by a GUI.

Setup Remote Access

After flashing the SD card, an initial setup is required. Within this setup, a static IP configuration has been implemented to enable remote connections, making the development process more convenient.

Overclocking and Cooling

The RPi4 has been selected as the development platform due to its low power consumption and small form factor while being capable of delivering relatively high computational performance. However, overclocking the CPU and GPU at frequencies of up to 2350 MHz and 800 MHz respectively. Because 2350 MHz is a very high frequency, it causes the CPU to overheat, resulting in frequency drops and system freezing. Therefore, a more stable frequency is recommended. A safety margin of 150 MHz is applied and the frequency is set at 2200 MHz. It is important to note that overclocking requires a cooling mechanism to keep the temperatures at an operational level. In this case, a heat sink and a fan were used. If it were for a more effective cooling mechanism, such as water cooling, it would be possible for the frequency to be higher. However, the downside of water cooling is its increased weight and space requirements, which make it unsuitable for portable devices.

Installation of ROS

The installation of ROS has been carried out following the instructions available on its official website. Specifically, the ROS-Base version has been installed, which includes the essential components and packages of ROS. It is noteworthy that two other versions, namely the Desktop Install and the Desktop-Full Install, are available, but they contain GUI tools. The reason for not selecting any of the GUI versions was because the OS does not support graphics, and to reduce the computational demand of the system. A beneficial package is the *web_video_server*, which provides the capability to send the video stream through the network, making it more comfortable for testing. Another package that was installed is the *rosbridge* package, which is responsible for the connectivity between the ROS and the JS program.

Download and Installation of RealSense SDK

RealSense SDK is available for ROS integration, and it was successfully downloaded and installed. As part of the integration process, the Infrared (IR) projector of the D435 was disabled by modifying some files of this tool. In addition, by adjusting certain parameters within the tool's files, several configurations were made. Some of these configurations included setting the cameras' resolution and frame rate to [848x480] and 30Hz, respectively. The goal was to maintain 30Hz and get the highest possible resolution for this frame rate. After conducting several tests, it was determined that the resolution mentioned was the optimal choice. It's worth noting that increasing the resolution beyond this point would have required more computational resources, resulting in frame drops. Regarding the T265, it was used for its IMU. The SDK provides a *launch* file that posts a topic for the accelerometer and a topic for the gyroscope, and they have a maximum rate of 62.5Hz and 200Hz respectively. Initially, a node was created to merge these two topics, but after doing research, it was discovered that a parameter existed that used linear interpolation to merge the data. The new topic has a rate of 200Hz and contains values from both the accelerometer and the gyroscope. This integration of the RealSense cameras with ROS has allowed for effective data collection and processing.

Installation of Requirements and ORB-SLAM3

Prior to installing ORB-SLAM3, it is essential to have several required packages installed. Initially, version 4.2 of the OpenCV toolset was downloaded and built from source, followed by the Pangolin library. As far as OpenCV is concerned, it had to be built from source due to the need of a specific version and it was necessary to configure some of the files before performing the build. After installing the prerequisites, ORB-SLAM3 was downloaded, and some modifications to the installation files were made in order to be able to be built. After the algorithm was built, ORB-SLAM3 for ROS followed. During the installation process, all CPU cores were utilized to reduce the time needed for this already time-consuming process, with the exception of when building the ORB-SLAM3 algorithm. In this case, the build required more RAM than the RPi4 had available, leading to the cores waiting for data that were not in the memory, making the whole system freeze. This could be solved by utilizing three of the four available cores. By doing this, the build was slower but did not result in freezing the OS. Another solution to

this problem was to add some gigabytes of swap memory if possible. When using swap memory, the OS uses the hard disk-like memory. Despite the significantly slower speed of the hard disk than the RAM, not to mention the SD card the system uses, it prevented the OS from freezing.

5.2 Component Calibration

In order to use the IMU device in the ORB-SLAM3 algorithm, it is crucial to calculate certain parameters. The method for calculating these parameters is described in [42, 43] and involves obtaining the necessary parameters for both the IMU and the camera-IMU connection. This method outlines the necessary steps to be followed to obtain these parameters. For this thesis, various bash script files were created that consist of a combination of the previous two, alongside custom configurations, in order to automatize the procedure. The first step in the process is to record a *rosbag* file. This recording should be no less than three hours long, and the IMU must be situated on a level surface. The recording should contain topics concerning the IMU (accelerometer, gyroscope) only. The second step is to compute the IMU parameters, using a tool called Kalibr [44] which is based on [45, 46, 47, 48]. These parameters include the accelerometer random walk, accelerometer noise density, gyroscope random walk, and gyroscope noise density. Random walk simulates thermo-mechanical noise error. It is referred to as a walk because it involves a series of steps, and random because each step is of a random size and direction. The noise density corresponds to the constant bias error. This error is calculated as the average of the values of the accelerometer or the gyroscope when there is no movement. These four parameters must be calculated in order to eliminate the corresponding errors. The third step is to record a *rosbag* file by following the instructions of Allan Variance ROS [49], which is based on [50, 51, 52]. The recorded *rosbag* must contain topics relating to the IMU (accelerometer, gyroscope) and the camera (left and right sensor). Additionally, specific movements must be performed in front of a target with a predetermined number of markers from the ArUco[53] library. The fourth and final step is to establish the connections between the poses of the left camera, the right camera, and the IMU in 3D space. These connections describe the linkage between devices in the form of extrinsic parameters, which need to be calculated. These parameters describe the relations of poses in 3D space. The symbols W, B, $C1$, and $C2$

are defined as frames of the World, Body, left camera, and right camera respectively. The Body is matched with the IMU, and its relation to the World is described as T_{WB}. The objective of this step is to calculate the transforms T_{BC_1}, and $T_{C_1C_2}$, which ORB-SLAM3 requires, to use them in Stereo-Inertial configuration. However, for the Stereo configuration of ORB-SLAM3, only the $T_{C_1C_2}$ is necessary. ORB-SLAM3 matches the Body to the left camera instead of the IMU. All of the previously computed parameters are passed to the ORB-SLAM3 algorithm in a ".yaml" file format.

5.3 ROS Implementation

5.3.1 Transform Library

In ORB-SLAM3, the *"map"* is created according to the first frame of the camera, resulting in a counterintuitive positioning system axes. This happens because the RealSense D435 camera complies with the ROS convention for the camera model[54] and places itself as shown in Figure 5.1. The colored axes depict the x-axis with red, the y-axis with green, and the z-axis with blue. However, the convention for a robot body in ROS is to have the forward, left, and up movement as the x, y, and z axes respectively, as shown in Figure 5.2. This leads to the need for modifying the camera's transform orientation.

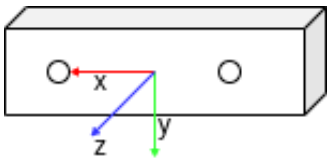


FIGURE 5.1: Transform of RealSense D435 according to the pinhole convention of OpenCV in RViz

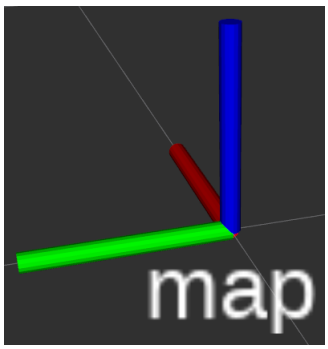


FIGURE 5.2: Transform of map in RViz

The transform tree represents all the relationships between the transform frames. In this thesis's case, the transform tree is the following. The first frame is the *"map"*, it is a world-fixed frame with the upward direction depicted by the z-axis, and it is created by the first frame of the camera. The reason that the system should always be powered on in horizontal orientation is that all the other transforms are based on the first frame of the camera and the creation of the *"map"*. The *"map"* is connected to the *"odom"* transform by ROS convention[55], while the *"realsense_optical"* is connected to the *"odom"* frame. This transform is dynamically calculated each time ORB-SLAM3 produces the coordinates and quaternion, and follows the ROS camera convention[54] with the *"_optical"* suffix. This means that the axes have the following orientation. The forward, right, and down movements are represented by the z, x, and y axes, respectively, as shown in Figure 5.1. However, since the body of the device should be represented as described before, the *"realsense_optical"* transform is followed by the *"realsense"* transform, which has the correct orientation. The *"realsense"* transform is static and contains a rotation of 135 degrees on the x-axis, 90 degrees on the z-axis, and 0 degrees on the y-axis, ensuring the correct orientation of the camera. Finally, the *"nail"* transform is calculated in the *nail_publisher* node as follows. Initially, the transform from *"map"* to *"realsense"* is determined, after which a translation is applied on the x-axis by 8.5 centimeters backward and on the z-axis by -*length*. The 8.5 centimeters backward represents the position of the RealSense from the center of the pole, and the *length* represents the pole's length, which can be changed even during runtime. The tree can be seen in Figure 5.3.

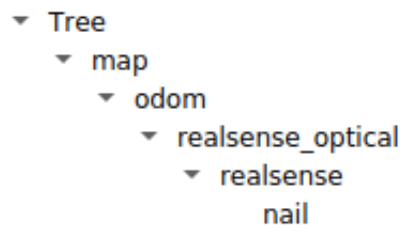


FIGURE 5.3: Transform tree in RViz

5.3.2 Launch Files

Launch files have been created for all the external devices used, as well as for organizing the overall functionality.

Top-Level Module

Two of the most significant, as far as organizing is concerned, *launch* files are the *system.launch* and *system-bag.launch*. Both of these files are responsible for initiating all the other essential *launch* files, and as a result, all the nodes required for the system to function. The *system.launch* file is essential, as it is responsible for initiating several critical nodes, including the *rs_d435.launch* file which initiates the RealSense camera, the *nail.launch* file which is described in detail in 5.3.2, the *rosbridge_websocket.launch* file, and the *slam_stereo.launch* file, as shown in Figure 5.5. *System-bag.launch* starts one more *launch* file than the *system.launch*, because it was developed to work with *rosbag* files for testing purposes. This extra file is a node named *web_video_server*, which provides the capability to watch a camera stream via a local server. A depiction of the contents of this node is shown in Figure 5.4. It is important to note that another *launch* file initiated by *system.launch* and *system-bag.launch*. This was the file that controlled the IMU, which was later removed for the reasons described in 6.2.

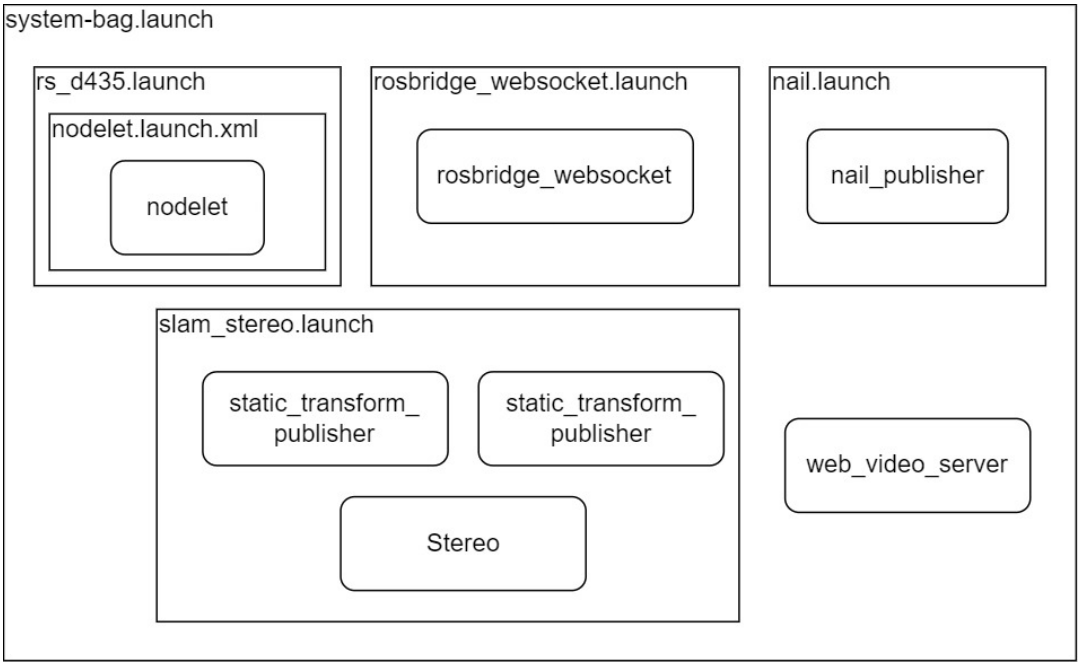


FIGURE 5.4: Depiction of launch file of the testing implementation

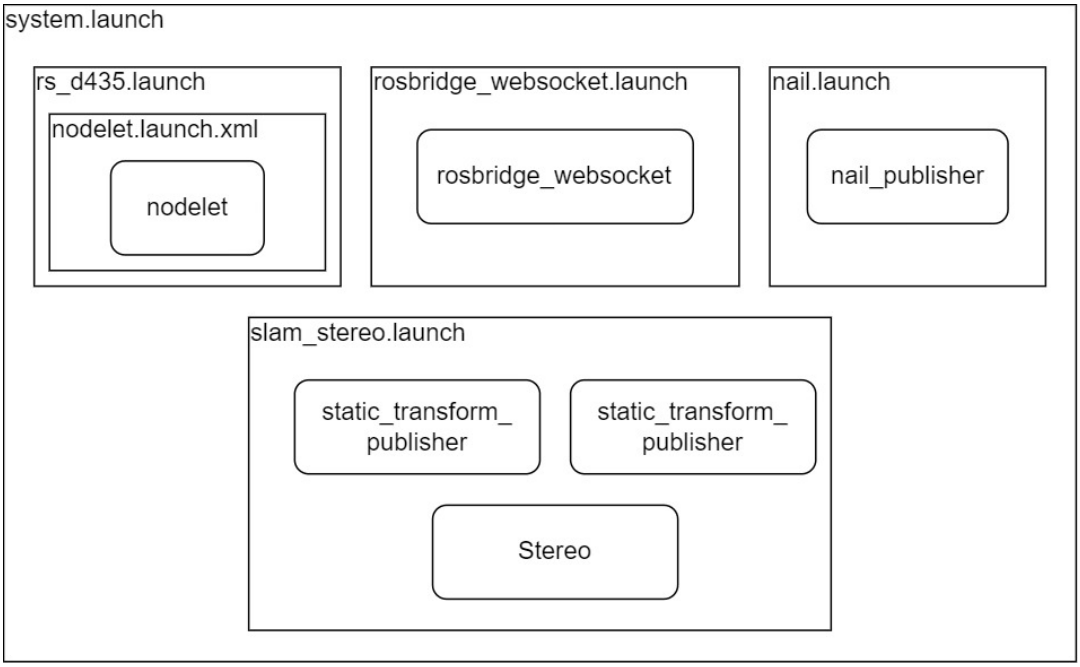


FIGURE 5.5: Depiction of launch file of the final implementation

Pixhawk IMU Launch File

One of the first *.launch* files created was used for the IMU of Pixhawk. To send data, a node needs to be running and a *rosservice* call needs to be executed to start the stream. To achieve this, a *.launch* file is created that starts the node by providing the path of the serial stream as a parameter. Additionally, the *rosservice* call *setstreamrate* is executed. This call specifies which stream should start and at what rate. A visual representation of this *.launch* file is shown in Figure 5.6.

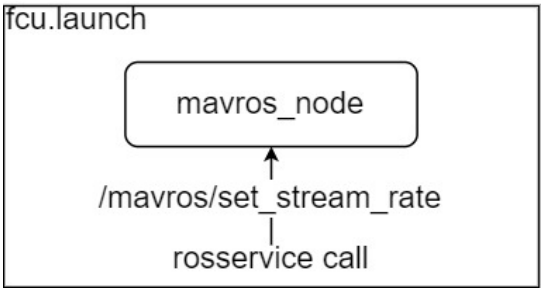


FIGURE 5.6: Depiction of the launch file of Pixhawk

RealSense Launch Files

The RealSense SDK comes with *launch* files that serve as examples. These files pass all the necessary parameters to the node, which then executes them. For the RealSense D435 and RealSense T265, the appropriate parameters have been configured. Specifically, for the T265, the *launch* files have the capability to disable the camera sensors from the parameters *enable_infra1* and *enable_infra2*, while enabling the IMU data, which is controlled by the *enable_gyro* and *enable_accel* parameters. Furthermore, the publishing rates for the accelerometer and gyroscope were set to the maximum rate of 62 Hz and 200 Hz, respectively. When the accelerometer and gyroscope data were combined, the rate was set to 200 Hz, and the unity method is linear interpolation. As for the RealSense D435, the emitter, depth, and RGB streams were disabled, and only the *enable_infra1* and *enable_infra2* parameters have been selected to be enabled, which control the two sensors of the stereo camera. Also, the rate was set at 30 Hz. Additionally, another *launch* file has been created to operate both the D435 and T265 at the same time. It is worth noting that the *launch* files for the RealSense cameras mentioned before are launching another *launch* file, which initiates the node responsible for controlling the devices as shown in Figure 5.7.

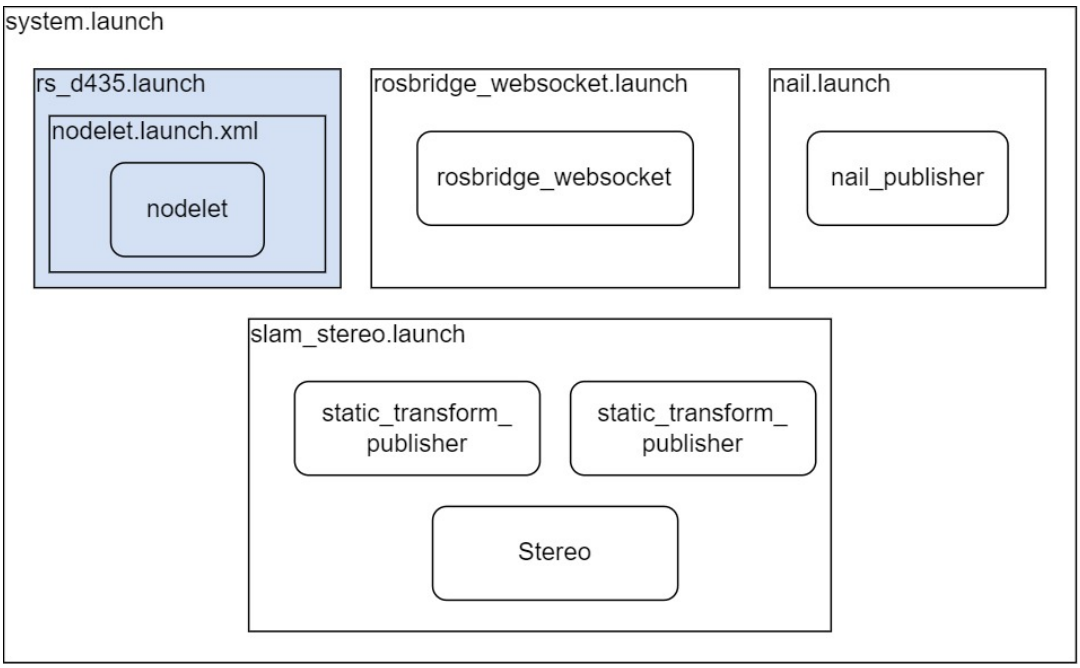


FIGURE 5.7: Launch file of RealSense D435 (in blue) in final system configuration

GUI Connection

The *rosbridge_websocket.launch* initiates the node that executes the functionalities to connect the main ROS system with the GUI application.

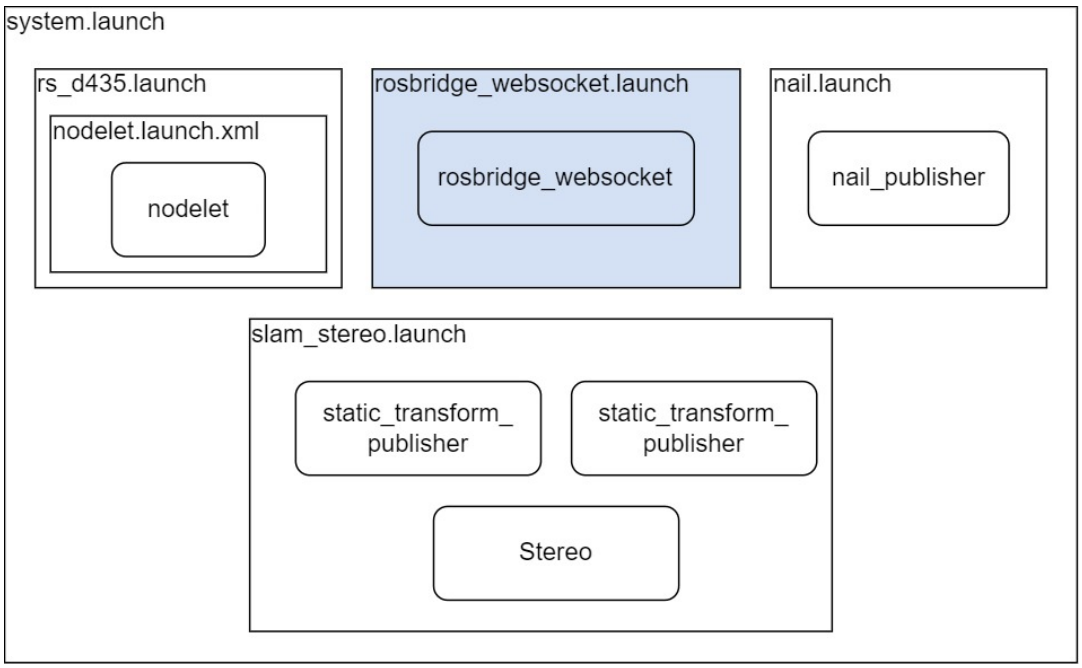


FIGURE 5.8: Launch file of Rosbridge (in blue) in final system configuration

Stereo Inertial Configuration Launch File

The *slam_stereo_imu_all.launch* file is designed to assist in organizing the way nodes are started and transforms are declared. Specifically, two static transforms are declared utilizing the *tf2_ros* package, namely the transformation between *"map"* to *"odom"* and the one between *"realsense_optical"* to *"realsense"*. Additionally, the launch file initializes three nodes, the *mavros_accel_gyro_separator*, *t265_accel_gyro_merger*, and *Stereo_Inertial*. The two first were used for testing purposes and the last one was used to run the node that executes ORB-SLAM3. A visual depiction is provided below.

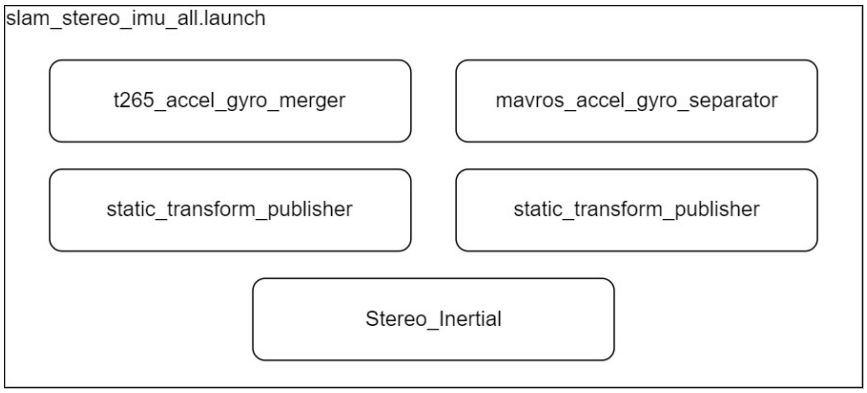


FIGURE 5.9: Launch file of Stereo Inertial configuration

Stereo and RGBD Configuration Launch File

The *slam_stereo.launch* and *slam_rgbd.launch* files initiate the same nodes for the static transform as those mentioned in 5.3.2. However, it is worth noting that the nodes for the IMU are not initiated here, as they are not used for these particular configurations. Instead of starting the *Stereo_Inertial* node, each *launch* file starts the *RGBD* node and the *Stereo* node of ORB-SLAM3 respectively. For the final configuration, the *Stereo* node has been chosen, as seen in Figure 5.10.

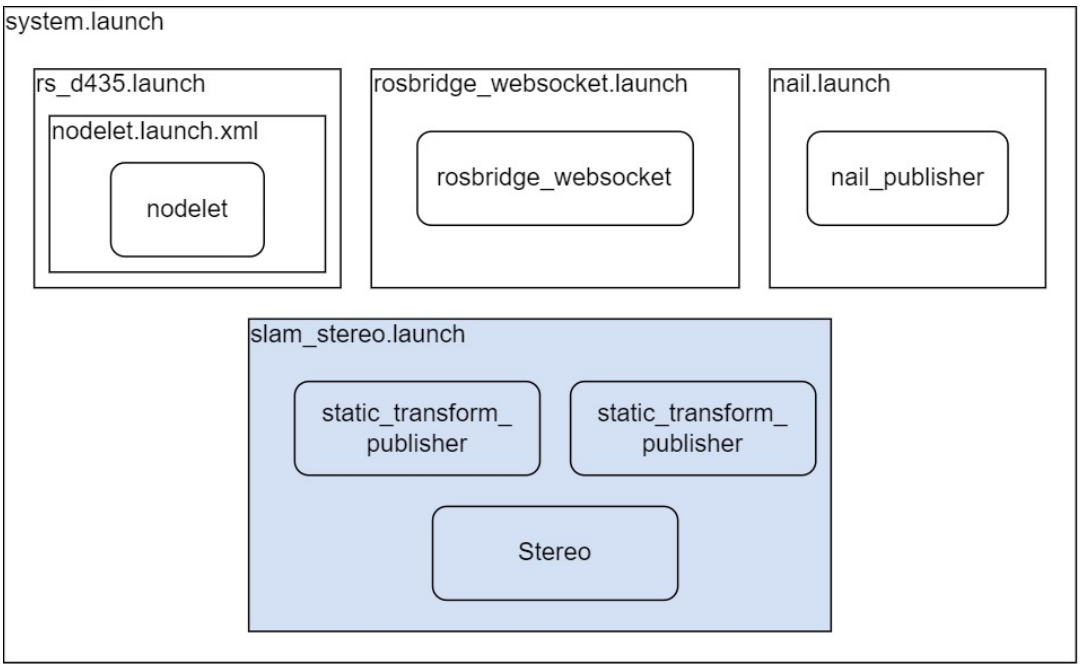


FIGURE 5.10: Launch file of Stereo Inertial (in blue) in final system configuration

Coordinate Extraction Launch File

The final *launch* file is *nail.launch*, which is responsible for initiating the *nail_publisher* node. This file, although it starts only one node, is responsible for parsing the default length of the pole as a parameter.

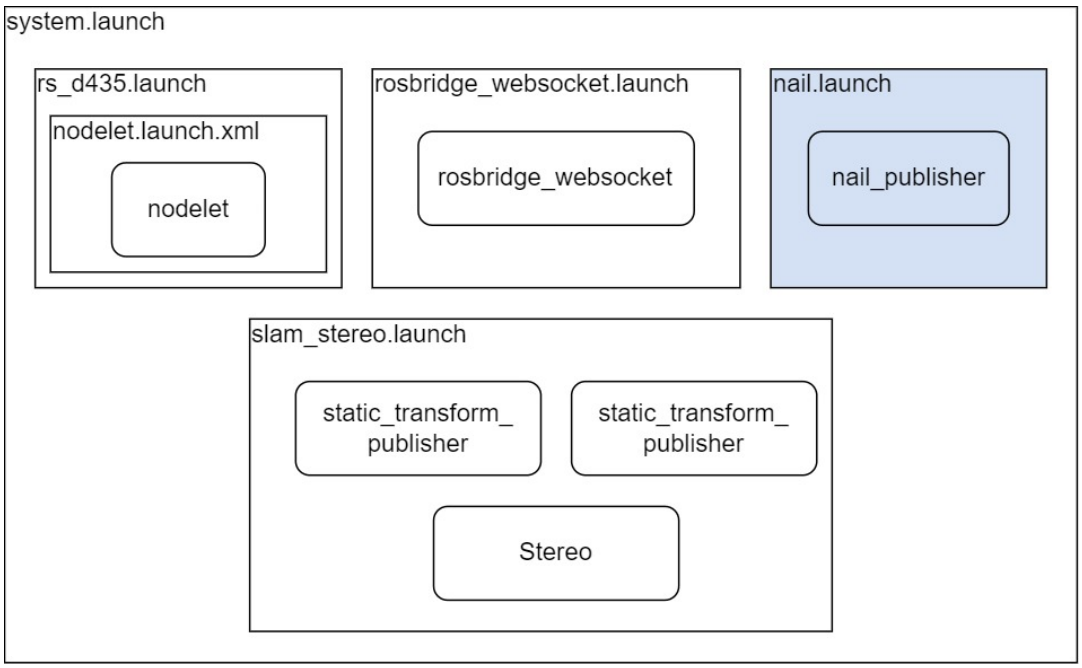


FIGURE 5.11: Launch file responsible for the transform calculation (in blue) in final system configuration

5.3.3 Nodes

The nodes that have been modified from the ORB-SLAM3, such as the ones that have been created for this thesis are presented here.

Separation of Accelerometer and Gyroscope in Pixhawk

Pixhawk is managed from a node named *mavros_node* from the package *mavros*, which publishes IMU data using the topic named */mavros/imu/data*. The message type is *sensor_msgs/Imu*[56], it is in a standardized format and its contents are shown below.

```
Header header
geometry_msgs/Quaternion orientation
float64[9] orientation_covariance # Row major about x, y, z axes
geometry_msgs/Vector3 angular_velocity
float64[9] angular_velocity_covariance # Row major about x, y, z axes
geometry_msgs/Vector3 linear_acceleration
float64[9] linear_acceleration_covariance # Row major x, y z
```

The necessity to separate the data from */mavros/imu/data* into two different topics, the */mavros/imu/accel* and */mavros/imu/gyro*, resulted in the need for a

node to be created. This node was named *mavros_accel_gyro_separator*. The first topic, */mavros/imu/accel*, is created by keeping only the values that are related to the accelerometer. These data are represented in the fields *linear_acceleration* and *linear_acceleration_covariance* from the *sensor_msgs/Imu* mentioned before. The other fields of the message are filled with zeros. Respectively, and as the name of the topic suggests, in the message of */mavros/imu/gyro*, only the data regarding the gyroscope are kept. These are the fields *angular_velocity* and *angular_velocity_covariance*, while the other fields are filled with zeros.

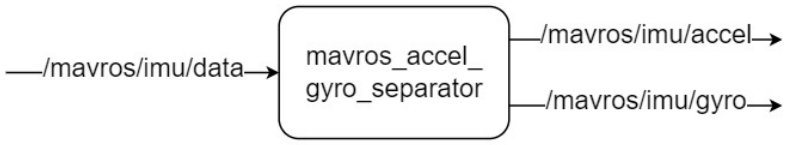


FIGURE 5.12: mavros_accel_gyro_separator

Synchronization of Accelerometer and Gyroscope in T265

While using the IMU of RealSense T265, the need to combine the accelerometer with the gyroscope data emerged. For this purpose, a node named *t265_accel_gyro_merger* was created. Its functionality is to receive messages from two different topics, at different rates, merge them, and publish the combined data, as shown in Figure 5.13. The two topics are */camera_t265/accel/sample* and */camera_t265/gyro/sample*, and they both contain the same type of message which is *sensor_msgs/Imu*. These topics are being published by a node provided by the RealSense SDK. From the two topics, the */camera_t265/accel/sample* contains information only about the accelerometer, so only the *linear_acceleration* and *linear_acceleration_covariance* fields contain values. The other fields contain zeros. Respectively, the */camera_t265/gyro/sample* contains information only about the gyroscope, and only the *angular_velocity* and *angular_velocity_covariance* contain values, with the other fields being zero. The produced message includes information about both the accelerometer and the gyroscope. After combining the messages, the new message is published via a topic named */camera_t265/imu* at the rate of the slowest receiving topic. In this thesis' case, this is the */camera_t265/accel/sample* with a rate of 62.5 Hz even though the */camera_t265/gyro/sample* sends data with a rate of 200 Hz. A higher rate can be achieved by repeating the last value between two continuously received messages.

FIGURE 5.13: t265_accel_gyro_merger

Stereo Inertial Configuration

Several nodes have been implemented by the ORB-SLAM3 developers with *Stereo_Inertial* being one of them. This node is responsible for the calculation of the output of the algorithm which is passed through the *track* variable. The *track* variable type is Sophus SE3f, which is a variable type that includes a quaternion and a translation consisting of four and three floats, respectively. However, the *track* variable refers to the center of the body as seen from the algorithm, which is the center of the IMU as far as the *Stereo_Inertial* node is concerned. To calculate the *track* variable, a series of steps are required. Firstly, three subscribers are created, one for the IMU and two for each stream of the stereo camera. Once a message from any of these topics arrives, it is stored in a buffer that is unique to each topic. Meanwhile, a new thread starts a function named *SyncWithImu*, where the synchronization between the images and the IMU data takes place, along with the calculation of the algorithm. An addition that was made in this node, is the conversion of *track* from SE3f to *geometry_msgs/TransformStamped*[57] type. This type has fields as shown below.

```
Header header
string child_frame_id
Transform transform
```

The *geometry_msgs/Transform*[58] has the following form

```
Vector3 translation
Quaternion rotation
```

This conversion is necessary for transmitting the transform and managing the transform tree using the "tf" package. Also, the appropriate values for the transform tree are assigned.

More specifically, the created transform has as *frame_id* the value *"odom"* and as *child_frame_id* the value *"realsense_optical"*. This means that the necessary rotations and translations between these two frames are provided. While

the total transform tree can be seen in Figure 5.3. Once the conversion and assignment are complete, the *TransformStamped* variable is ready to be broadcast. This is achieved using the "tf2_ros" package. By broadcasting the transform, it is integrated with the transform tree, and the values can be monitored in the */tf* topic.

Stereo Configuration

The *Stereo* node is another node that has been adapted for the requirements of this thesis and is developed by the ORB-SLAM3 developers. Its primary function is the same as 5.3.3, meaning to facilitate the computation of the output of the ORB-SLAM3 algorithm. However, it differs in that it uses two images without the IMU. Additionally, the node's message synchronization is performed using ROS functions instead of custom functions. However, the *track* defined and used in the 5.3.3 is also utilized in this node. The additions that were made from the provided node and are identical to the ones at 5.3.3 are the following. These include the conversion of *track* from SE3f to *geometry_msgs/TransformStamped* and its broadcast. Before the conversion, the transform is multiplied by a transform that brings it into the pose seen in Figure 5.1 in relation to the map. Moreover, a *rosservice* server has been added, which is responsible for setting the algorithm to perform only localization. This is done to increase the frame rate of the output. It is worth mentioning that this is the node utilized in the final system.

RGBD Configuration

The last node that was adapted for the purposes of this thesis and was originally created by the ORB-SLAM3 creators, is the *RGBD* node. It is responsible for the *RGBD* configuration of the algorithm and has the same functionality as the 5.3.3, with the only difference being the topics used. Specifically, in this node, the topics are the */camera/color/image_raw* that contains messages from the RGB camera, and the */camera/depth/image_rect_raw* that contains messages for the depth stream. The output of the algorithm is then calculated and the same transformation is applied. However, the *rosservice* server is not used in this implementation, as it was an attempt to increase the output rate of the algorithm. Therefore, it was unnecessary to modify it to its latest version.

Nail Publisher

The coordinates calculated from the algorithm are contained in the *"realsense_optical"* transform, but there are two more transformations required to get to the nail transform, which contains the final coordinates. These transformations are *"realsense"* and *"nail"*. The *"realsense"* is a static transform declared in the *.launch* file that starts the *Stereo* node. The *"nail"* transform is calculated in the *nail_publisher*, which is a publisher node, as follows. Initially, a *rosservice* server is created. This server is responsible for getting the length of the pole and assigning it to the transform. A default value of 1.5 meters has been set. After that, the name of the published topic is defined to be */nail_publisher/coordinates*, and the published rate is set at 30 Hz. Next, a search for the transform from *"map"* to *"realsense"* frame takes place, followed by the creation of a new transform referred to as the relation from *"realsense"* to *"nail"*. The newly created transform contains information on the distances between the body, as seen from the algorithm, and the end of the pole. The body, which is the RealSense stereo camera is 8.5 centimeters forward of the center of the pole and 1.5 meters up of the end of the pole. However, this is not always the case. The user can change the length of the pole at any time and then set the new value through the application. This value is passed to the transform from *"realsense"* to *"nail"*. After that, the two transforms are multiplied to get the final transform. From this transform, only the x, y, and z values are published because only the coordinates are needed and not the orientation of the device.

5.4 Chapter Summary

The implementation of the Raspberry Pi 4 as the development platform required careful attention to various configurations and setups. The selection of Ubuntu Server 20.04 as the operating system, implementation of remote access, and careful consideration of overclocking and cooling set the foundations to ensure the stability and performance of the system. The integration of ROS and RealSense SDK were critical steps that further enhanced the platform's capabilities, enabling effective data collection and processing. Additionally, the installation of required packages and ORB-SLAM3, as well as the algorithm itself was essential to the overall development.

Chapter 6

System Integration, Verification, and Performance Evaluation

In this chapter, the integration of the components is described, as well as the verification of the system and the performance evaluation. The verification of the system is conducted through a set of experiments, which were performed to demonstrate the accuracy of the final system in various locations and environments while running in real-time.

6.1 System Integration

The final system is an uncomplicated group of three devices. A smartphone or laptop, the RealSense D435 stereo camera, and the RPi4 where all the computations take place. The connection between the RPi4 and the stereo camera is achieved through USB 3.0, while the smartphone is connected via WiFi with the RPi4. The components are controlled using the ROS framework. The software components are shown in Figure 5.5, and are described in 5.3.2. Among the devices contained by the system was the IMU which ultimately, was not included. Its integration revealed the difficulties described in 6.2, where also the tried solutions are described. These challenges resulted in not containing any IMU device in the final system and concluded the final system as shown in the block diagram in Figure 6.1.

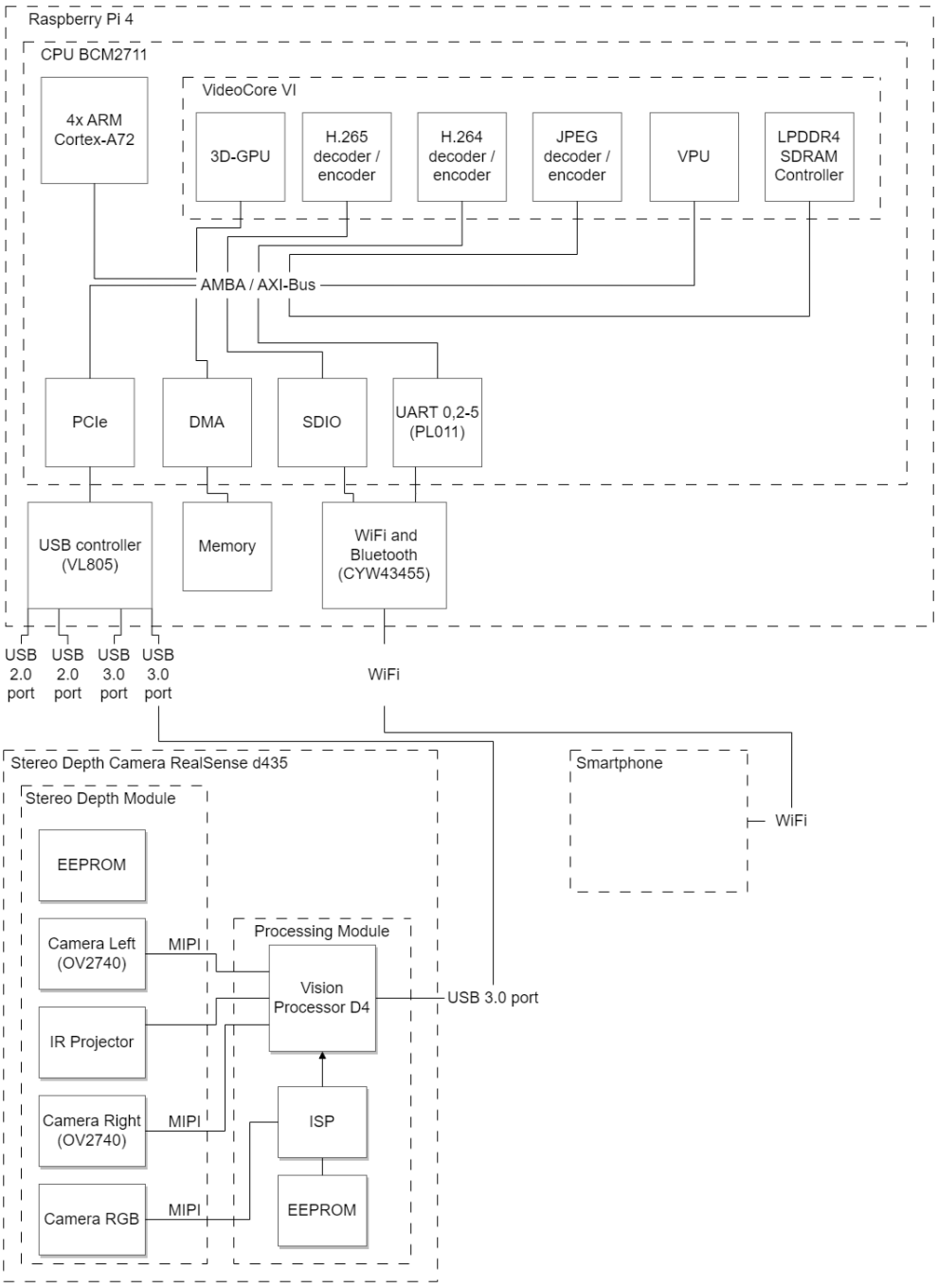


FIGURE 6.1: System block diagram using the Raspberry Pi 4 with the RealSense D435 as a stereo camera and a smartphone as a user interface.

6.1.1 3D Printed Case

During the process of integrating the physical components, the use of 3D printing was employed. The main objective was to design a mount that could securely hold the RPi4, the RealSense D435, and the power bank on the pole.

The initial design was created by Angelos Antonopoulos, and it features a modular layout that allows for easy removal or addition of components by simply unscrewing three bolts that secure them in place, as shown in Figure 6.2. However, this design did not include a base to hold the power bank or the RPi4, as shown in Figure 6.3, because the needs of the system were still unclear. The process of creating the casing includes the designing of the model and then printing it. In Figure 6.4 the designed models are shown. The second design included the aforementioned bases together with a mount for the D435 as seen in Figure 6.5. The printing material was acrylic styrene acrylonitrile (ASA), a type of plastic that combines durability, while keeping the cost at low levels.

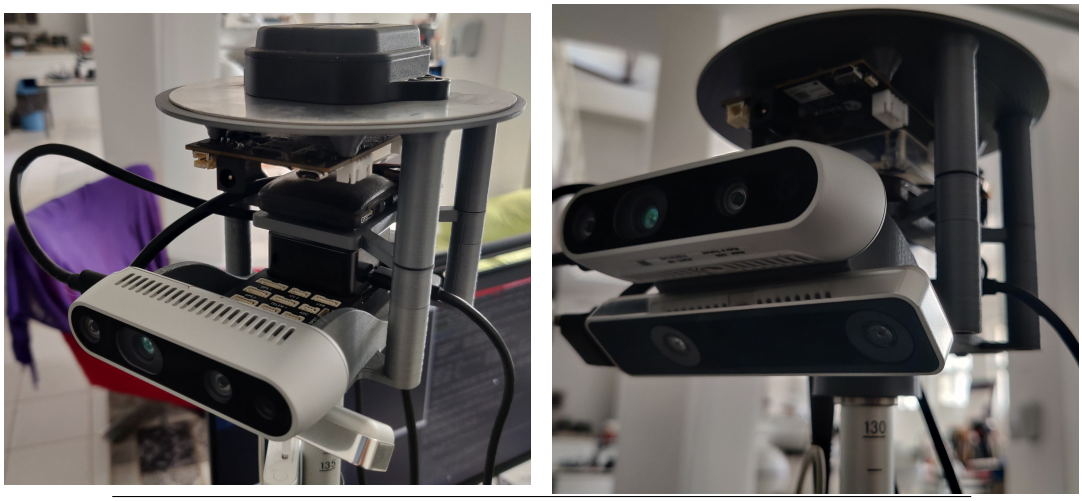
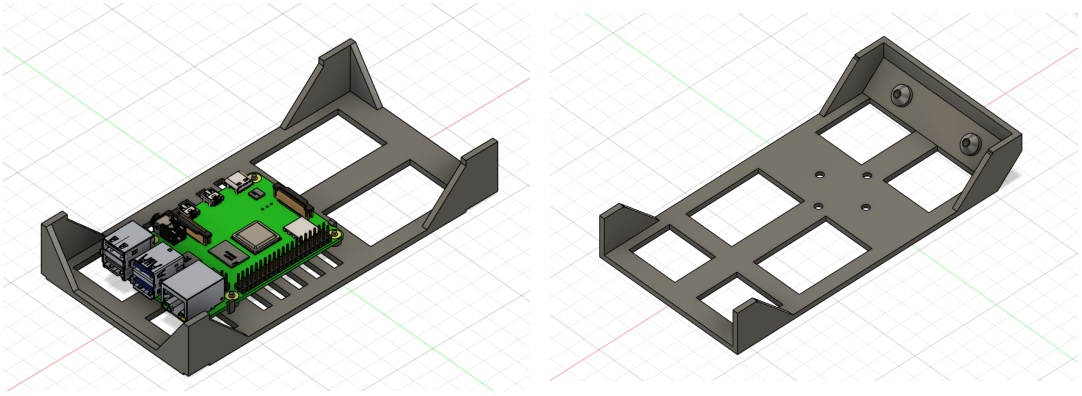


FIGURE 6.2: Initial mount of the components of the system.

FIGURE 6.3: Placing of the RPi4 and the power bank in the initial mount of the components of the system.



(A) 3D model for the casing of the Raspberry Pi.

(B) 3D model for the casing of the power bank and mount of the RealSense D435.

FIGURE 6.4: Designed 3D models for the casing of the system.

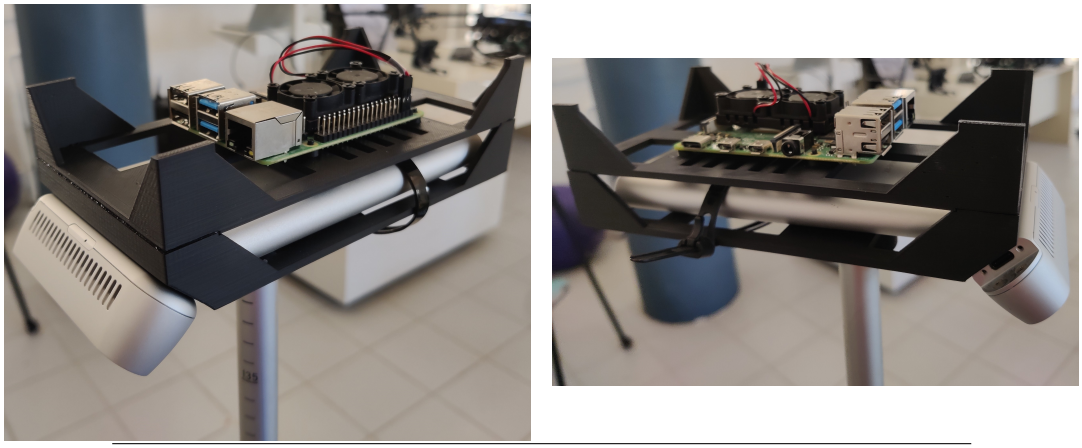


FIGURE 6.5: Final configuration of the system.

6.2 IMU Integration Issue

After everything was installed it was time to start working with the configurations of the algorithm. As mentioned before, ORB-SLAM3 can run as one of the following configurations: Monocular (single camera), Monocular Inertial, Stereo (two cameras), Stereo Inertial, and RGBD. When the configurations of ORB-SLAM3 for Monocular or Stereo were tested, the algorithm was running smoothly and only minor changes had to be made to make it work. In the case of Monocular Inertial or Stereo Inertial, the algorithm crashed a few seconds after starting. This indicates that there is an IMU-related issue. It is worth mentioning that, as a way to eliminate the possibility of device incompatibility another IMU was tested. Instead of using only the Pixhawk IMU, the IMU from RealSense T265 was also utilized. The possible causes were found to be four according to [1] and [59]. Below there is a description of them with some possible solutions.

6.2.1 IMU Rate

In [1] it is mentioned that in the configurations using the IMU, it is crucial for the IMU rate to be *"orders of magnitude higher than the frame rate"*. Since the frame rate was 30 Hz, the IMU rate was set at 300 Hz. In Pixhawk's IMU, the rate was set at 300 Hz, meanwhile, this was not the case for the T265. In the last, the maximum supported rate for the accelerometer is 62 Hz, and for the gyroscope is 200 Hz. A node was created as an attempt to interpolate the data and reach the rate at 200 Hz. Nevertheless, this solution did not work for any of the IMUs.

6.2.2 IMU Calibration

One more thing that could cause the IMU issue is miscalculated calibration parameters. They are calculated during the calibration procedure, as mentioned in 5.2. There is a step, in which the IMU must be on a level surface, and a *rosbag* file must be recorded with a duration of at least three hours. Then this *rosbag* file is processed to extract the intrinsic parameters. During this phase, the IMU must be completely still and in an empty room without anyone creating vibrations. However, if vibrations that will give a wrong calibration result are recorded, the recording must be repeated. Numerous recordings have been repeated for both IMUs and all of them at night time to ensure the validity of the recording. Nevertheless, it didn't provide an efficient solution to the problem.

6.2.3 Not Synchronized Messages

It is essential that the messages received by the IMU and the camera are synchronized. However, because they have different rates, the messages might arrive with a significant time difference. In ORB-SLAM3 this difference is checked and if it is higher than a predefined limit it prompts an error message. In the case that this happens continuously, the algorithm cannot continue to run and terminates. To ensure that this didn't happen, the timestamps of the IMU messages were compared to the timestamps of the camera messages. The difference they had was in the desired limits of the algorithm. An additional action was performed and this was to increase the limit that is set in the algorithm. However, even after performing this action, the problem was not solved.

6.2.4 IMU Initialization

The authors of [1] annotate that, the IMU initialization process requires at least two seconds to complete. During this time, the IMU must go through enough rotation and acceleration to be effective, but not so much that it causes frame losses from the camera. Despite numerous attempts to resolve the issue by testing faster and slower movements, it remains unresolved.

Eventually, the suggestion of the authors of [1] was followed, which is that in systems where the initialization fails or is difficult to achieve, the Mono and

Stereo configurations should be used instead of Mono-Inertial and Stereo-Inertial respectively.

6.3 Improving Output Rate

In this section, two attempts to improve the output rate of the algorithm will be described. In the first one, the RGBD stream from the RealSense D435 stereo camera is utilized, and in the second one, the ability of the algorithm to perform only localization is tested.

6.3.1 RGBD and Stereo Comparison

The Stereo-Inertial configuration was tested but not used due to the reasons described in 6.2. Besides the four configurations of the ORB-SLAM3, there is also the RGBD configuration. The RGBD and Stereo configurations were tested and the last one was used in the final system. A more detailed explanation follows. In the Stereo configuration, the depth is calculated using the images of the stereo camera according to the procedure discussed in 3. The depth calculation occurs for every frame resulting in increasing the computational needs in the RPi4 CPU. D435 provides a depth stream and as a way to avoid these calculations, this depth stream was exploited. The RGBD configuration utilizes the depth stream from the D435 with the stream from the RGB camera.

While running in the RPi4, ORB-SLAM3 Stereo produces results at a rate of approximately 8 Hz, therefore the rate of the RGBD configuration is expected to be higher as fewer computations occur. However, the rate was not increased from 8 Hz even with the RGBD configuration. This is explained because the RPi4 receives the depth data at a slower rate and the data is twice as much. The stereo camera images are in an 8-bit format, while the depth and RGB images are in a 16-bit format. There was an attempt to reduce the rate of the D435 from 30 Hz to 15 Hz without having a significant difference. In order to test the RGBD configuration, the RGBD node was used, as described in 5.3.3, as well as the launch file 5.3.2.

6.3.2 Localization Only Mode

One more attempt to improve the output rate from the algorithm relied on the reduction of computations the CPU has to execute. This attempt was to

exploit the ability of the algorithm to perform only the localization part, and the functionality is described in 5.3.3. The localization only mode was meant to be activated only after the loop closure had occurred and the map had been created. The results were promising as the output rate increased from 8 Hz to 14 Hz. However, it came with a drawback that made it unusable. A few seconds after it was activated, the estimation of the position of the device began to indicate a drift error, that as time passed it was getting worse until the error was extremely high.

6.4 Performance Evaluation

While the experiments were conducted, it was noticed that if the pole mapped the area by "looking" at the floor, the results presented a noticeable improvement. This is expected for the reason that when viewing the floor, more features are provided at a smaller distance than features provided by the walls. Therefore, a new arrangement of the RealSense has been performed. The initial placing was facing forward, meanwhile, its new position is at a 45°angle facing downwards.

As discussed in section 2, when it comes to GNSS-denied environments for surveying, measuring tape, total station, 3D laser scanners, and laser distance meters are considered some of the most reliable tools, while the total station is one of the most accurate options. The measuring tape was used as a reference for the experiments. It is worth mentioning that, the performance of algorithms for SLAM is generally measured using Absolute Trajectory Error (ATE) and Relative Position Error (RPE) metrics. However, in this thesis system, the primary concern is not the trajectory or the orientation of the system but the coordinates that derive from them. Therefore, the accuracy of the system is evaluated using the distance error as an accuracy metric. Additionally, the mean absolute error (MAE) is taken into consideration.

6.4.1 Point-Capturing Procedure and Post-Processing Results

The process of point-capturing involves a series of steps. Initially, the system must be powered on, and a smartphone must be connected via WI-Fi and enter the web application. Secondly, the path that the user must follow

should be in a loop-like shape containing the area of interest. This is important for the algorithm to complete the loop closure step. The user's movement should be such that the system can recognize as many features as possible. Finally, the end of the pole should be placed at the desired point, and the point should be captured using the smartphone. This simple procedure could be challenging because the processing power of the RPi4 is insufficient to provide the system with a high output rate. Therefore, the movement must be slow, especially when performing a turn, to ensure that the system can provide accurate results and avoid frame drops.

Initially, a set of points were marked on the ground and their distances were measured using a measuring tape so they would be used as ground truth. After that, point-capturing was performed following the aforementioned procedure. The coordinates of the captured points were downloaded, from the web application into a file for further evaluation. After that, the captured points were processed by a MATLAB script that was developed to compute the distances between the points.

Post-Processing and Error Calculation

The developed MATLAB script imports the captured points from the downloaded file and calculates the distances between them by using the Euclidean distance formula, which is then compared with the measured distances. The error, in centimeters, is calculated by this comparison and later computed in percentage accuracy. The experiments are organized in two phases, which will be described later. As far as the first phase is concerned, one of the points was selected as a zero point, with coordinates (0, 0, 0) to establish a reference point. The script was designed to transform all points to the zero point by subtracting their values for each axis. For example, in Figure 6.6, the *p2* point was chosen as the zero point, and its coordinates were subtracted from all the other points. To get a better understanding, the MATLAB file was improved to display the captured points, along with the ground truth points. Additionally, the perimeter of the captured and ground truth points was also displayed as shown in Figure 6.6. Regarding the second phase, a new script was developed that calculated the error, in centimeters and the percentage accuracy similarly to the first phase. Additionally, it was updated to calculate a mean accuracy value by using measurements acquired by capturing points in the same area multiple times.

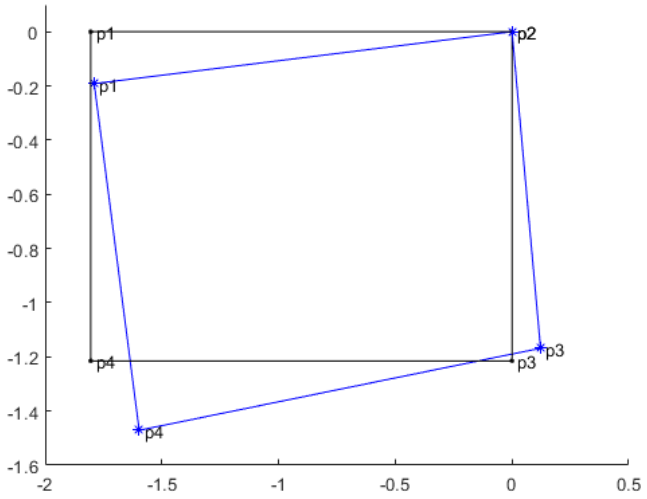


FIGURE 6.6: Points acquired by following a rectangular path. The black and blue points and perimeter represent the measured distances and the estimated respectively.

6.5 Measurements and Results

6.5.1 Experiment Setup Description

The experiments were carried out in four distinct regions that were categorized based on ground texture, and their size. Regarding their ground texture, two of them are high-featured, while the other two consist of a more plain and simple texture.

The experiments are divided into two phases. In the first phase, the system performance and accuracy are generally examined in different environments and shapes. As far as the ground texture is concerned, the areas of interest are the inside of a lab and a part of an outdoor square for the high-featured and low-featured areas respectively. Regarding the dimensions, the small indoor area varies from about 1.2 meters to 2.9 meters, while the small square area is approximately from 2.1 meters to 2.5 meters. The distances of the large indoor area are between 0.7 meters to 4.4 meters, while the large square area varies from 4.25 meters to 4.9 meters.
In the second phase, the validity of the accuracy is evaluated by performing multiple times the point-capturing procedure, while maintaining the same area and the same points. The areas are the same as in the first phase as far as the texture is concerned, however, there are slight differences regarding

the dimensions. The small high-featured and low-featured areas are 1.3 x 2.8 meters and 1.5 x 3.0 meters respectively, while the large high and low-featured areas are 2.8 x 8.9 meters and 3.0 x 9.9 meters.
The small and large areas were chosen to explore whether the accuracy of the system is affected by the size of the interested area. While the high and low-featured areas were selected examine the impact the features have on the accuracy. It is expected to notice better results in small areas, and in high-featured areas.

6.5.2 Indoor Small Room Experiments

The first accuracy experiment included three points shaping a triangular. This shape was chosen for its simplicity, to initially test the distance differences as it includes three points and three distances connecting them. The path followed during the test and the points acquired are shown in Figure 6.7, while the estimated distances and points are shown in Figure 6.8. For this test, the path shown in Figure 6.7 was followed three times. The first time was for the algorithm to perform the loop closure step. The second and the third were to acquire the points. In Figure 6.8, the points displayed in green refer to the first point acquisition pass, and the points in blue to the second point acquisition pass. The actual measured distances, the distances from the acquired points, their differences, and the accuracy are shown in 6.1. The error is calculated as the difference between the actual and estimated distances. It varies from 7.1 to 14.4 cm for the first pass, and from 4.2 to 6.3 cm for the second pass, while the MAE is 10.8 cm, and 5.3 m respectively. It is important to note that the mean accuracy is 94.55% for the first pass and 97.24% for the second pass.

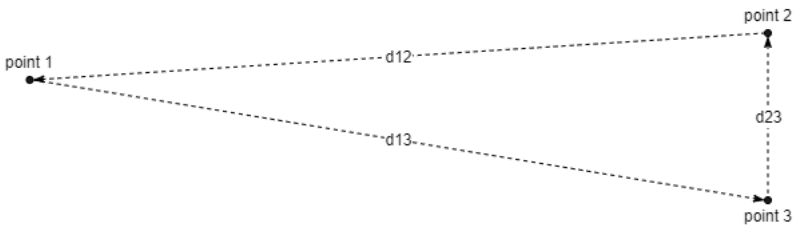


FIGURE 6.7: Path followed for testing and point acquisition. The area is a small room and the distances are d12 = 2.744 m, d13 = 2.880 m, and d23 = 1.215 m.

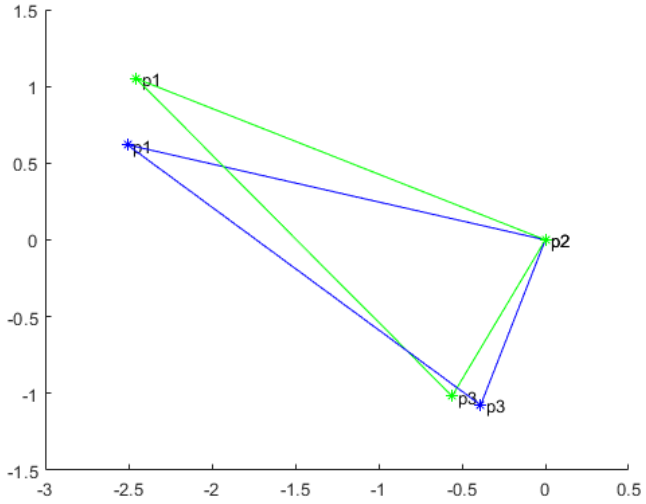


FIGURE 6.8: Points acquired by following the triangular path in Figure 6.7. The p1, p2, and p3 are point 1, point 2, and point 3 respectively. The blue points are taken during the first pass, and the green during the second.

Distance of points	Measured distance (m)	Estimated distance first pass (m)	Error (m)	Accuracy Percentage (%)	Estimated distance second pass (m)	Error (m)	Accuracy Percentage (%)
1-2	2.744	2.600	0.144	94.75	2.681	0.063	97.70
1-3	2.880	2.728	0.152	94.72	2.807	0.073	97.47
2-3	1.215	1.144	0.071	94.16	1.173	0.042	96.54

TABLE 6.1: Actual values, measurements, and errors from the path displayed in Figure 6.7. The MAE is 10.8 cm for the first pass, and 5.3 cm for the second pass.

For the next experiment, four points were captured, shaping a rectangular. This is also a simple shape that helps with understanding if there are any scaling issues. The path followed during the test and the points acquired are shown in Figure 6.9, while the estimated distances and points are shown in Figure 6.10. For this test, the path shown in Figure 6.9 was also followed three times for the same reasons as before. In Figure 6.10, the points displayed in green refer to the first point-capturing pass, and the points in blue to the second point-capturing pass. The actual measured distances, the distances from the acquired points, their differences, and the accuracy are shown in 6.2.

The error varies from 0.1 to 8.4 cm for the first pass, and from 3.9 to 7.2 cm for the second pass, while the MAE is 4.2 cm, and 5.7 cm respectively. Moreover, the mean accuracy is 96.85% for the first pass and 96.24% for the second pass.

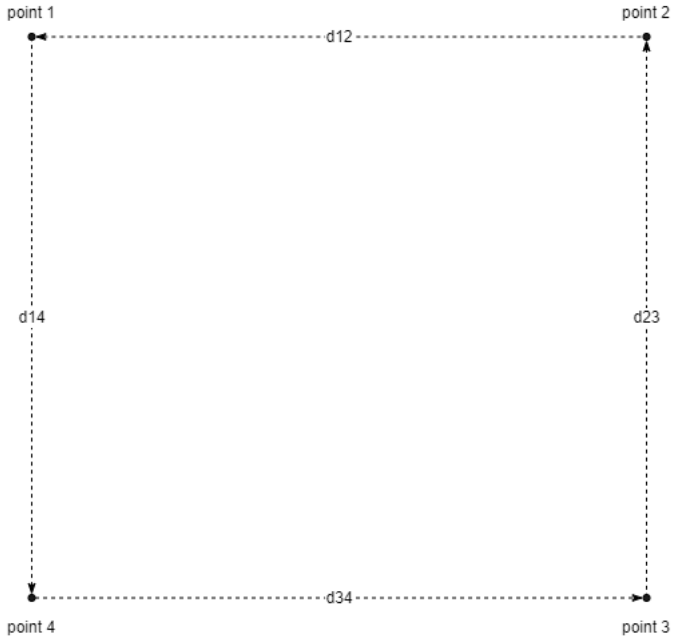


FIGURE 6.9: Path followed for testing and point acquisition. The area is a small room, the path represents a rectangular and the distances are d12 = 1.805 m, d23 = 1.215 m, d34 = 1.805 m, and d14 = 1.215 m.

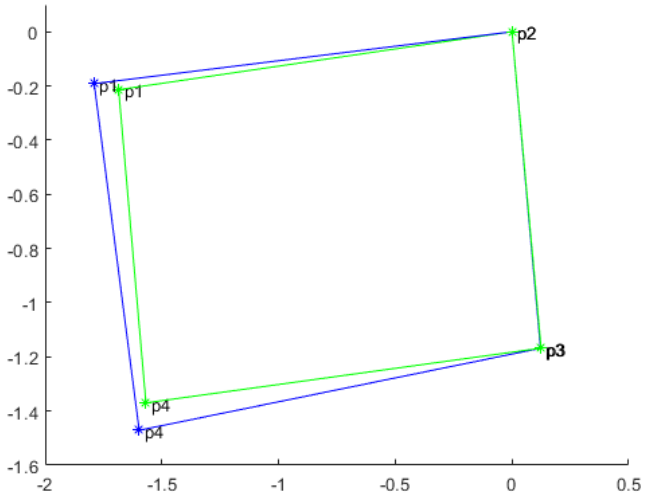


FIGURE 6.10: Points acquired by following the rectangular path in Figure 6.9. The p1, p2, p3, and p4 are point 1, point 2, point 3, and point 4 respectively. The blue points are taken during the first pass, and the green during the second.

Distance of points	Measured distance (m)	Estimated distance first pass (m)	Error (m)	Accuracy Percentage (%) (%)	Estimated distance second pass (m)	Error (m)	Accuracy Percentage (%)
1-2	1.805	1.804	0.001	99.94	1.736	0.069	96.18
2-3	1.215	1.175	0.040	96.71	1.176	0.039	96.79
1-4	1.215	1.299	0.084	93.09	1.166	0.049	95.97
3-4	1.805	1.763	0.042	97.67	1.733	0.072	96.01

TABLE 6.2: Actual values, measurements, and errors from the path displayed in Figure 6.9. The MAE is 4.2 cm for the first pass, and 5.7 cm for the second pass.

6.5.3 Outdoor Square Experiments

Continuing with the experiments, two of them took place in an outdoor area. Points were captured in a square by following a small and a large rectangular path for simplicity. The captured points, the distances, and the path followed are shown in Figure 6.11. The path had the same shape in both small and large areas, with the difference that in the large area, it was scaled up. The path was followed two times, the first was for the algorithm to perform the loop closure step and the second was for the point-capturing procedure. The

estimated distances for the small area are shown in Figure 6.12 and for the large area in Figure 6.13. The Tables 6.3, 6.4 present the actual, the estimated distances with the error of the estimated, and the accuracy. The errors vary from 1.3 to 17.9 cm for the small area, and from 33.9 to 43.6 cm for the large area, while the MAE is 9.3 cm, and 38.5 cm respectively. The mean accuracy is 95.98% for the small area and 91.59% for the large.

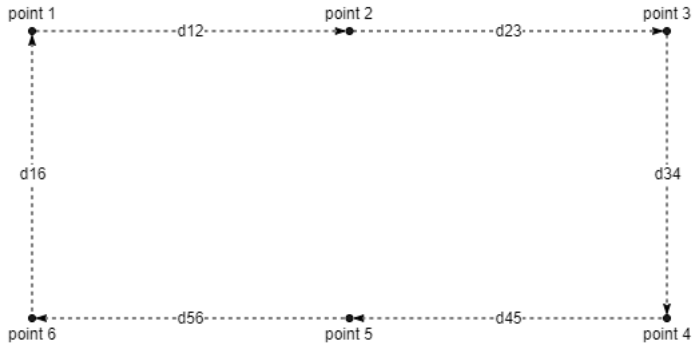


FIGURE 6.11: Path followed for testing and point acquisition. The area is a small and large outdoor square, the path represents a rectangular. The distances for the small area are d12 = 2.396 m, d23 = 2.438 m, d34 = 2.126 m, d45 = 2.436 m, d56 = 2.396 m, and d16 = 2.124 m. The distances for the large area are d12 = 4.834 m, d23 = 4.862 m, d34 = 4.268 m, d45 = 4.874 m, d56 = 4.834 m, and d16 = 4.270 m.

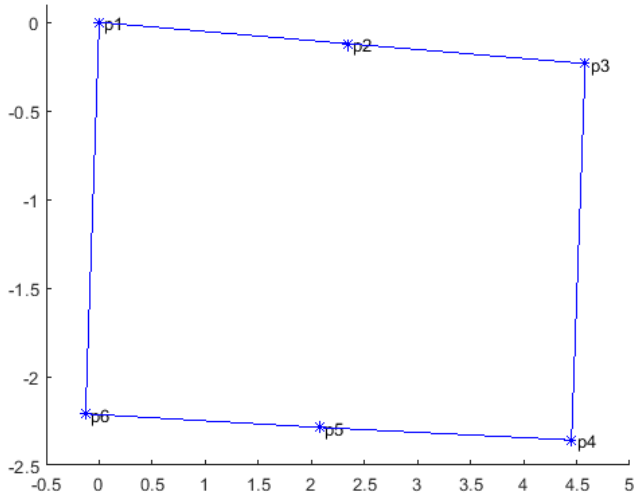


FIGURE 6.12: Points acquired at the small square by following the path in Figure 6.11. The p1, p2, p3, p4, p5, and p6 are point 1, point 2, point 3, point 4, point 5, and point 6 respectively.

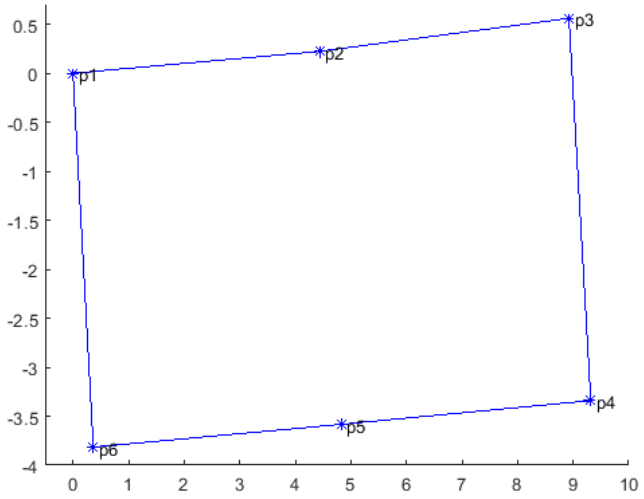


FIGURE 6.13: Points acquired at the large square by following the path in Figure 6.11. The p1, p2, p3, p4, p5, and p6 are point 1, point 2, point 3, point 4, point 5, and point 6 respectively.

Distance of points	Measured distance (m)	Estimated distance (m)	Error (m)	Accuracy Percentage (%)
1-2	2.396	2.366	0.030	98.75
2-3	2.438	2.269	0.169	93.07
3-4	2.126	2.139	0.013	99.39
2-5	2.122	2.205	0.083	96.09
4-5	2.436	2.388	0.048	98.03
5-6	2.396	2.217	0.179	92.53
1-6	2.124	2.252	0.128	93.97

TABLE 6.3: Actual values, measurements, and errors from the small square path displayed in Figure 6.11. The MAE is 9.3 cm.

Distance of points	Measured distance (m)	Estimated distance (m)	Error (m)	Accuracy Percentage (%)
1-2	4.834	4.453	0.381	92.12
2-3	4.862	4.496	0.366	92.47
3-4	4.268	3.929	0.339	92.06
2-5	4.268	3.832	0.436	89.78
4-5	4.874	4.504	0.370	92.41
5-6	4.834	4.464	0.370	92.35
1-6	4.270	3.839	0.431	89.91

TABLE 6.4: Actual values, measurements, and errors from the large square path displayed in Figure 6.11. The MAE is 38.5 cm.

6.5.4 Indoor Large Room Experiments

The following experiment was performed indoors, in a large area. The arrangement of the points is random as shown in Figure 6.14, where the path followed and the distances are also shown. The estimated distances are shown in Figure 6.15 in which the first pass is displayed with blue color and the second pass with green. The actual and the estimated distances, the error, and the accuracy are presented in Table 6.5. The errors vary from 0.2 cm to 38.3 cm for the first pass, and from 6.4 cm to 41.5 cm for the second pass, whereas the MAE is 16.2 cm, and 17.3 cm respectively. The mean accuracy is 90.76% and 89.49% for the first and second pass respectively.

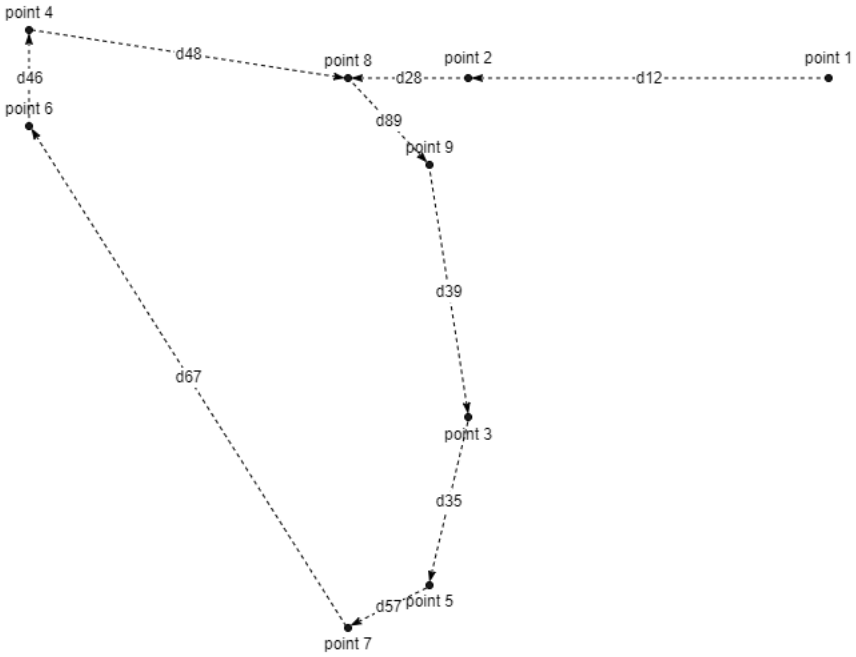


FIGURE 6.14: Path followed for testing and point acquisition. The area is a large room, and the points are randomly distributed in the area. The distances d12 = 2.733 m, d28 = 0.912 m, d89 = 0.864 m, d39 = 1.845 m, d35 = 1.253 m, d57 = 0.684 m, d67 = 4.345 m, d46 = 0.690 m, and d48 = 2.449 m.

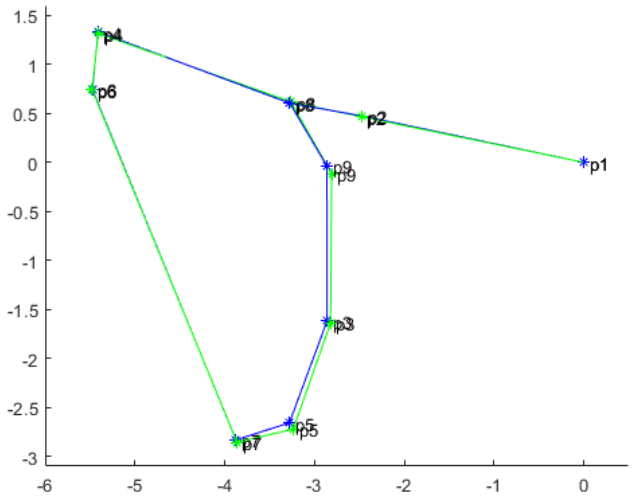


FIGURE 6.15: Points acquired at the large room by following the path in Figure 6.14. The p1, p2, p3, p4, p5, p6, p7, p8, and p9 are point 1, point 2, point 3, point 4, point 5, point 6, point 7, point 8, and point 9 respectively. The blue points are taken during the first pass, and the green during the second.

Distance of points	Measured distance (m)	Estimated distance first pass (m)	Error (m)	Accuracy Percentage (%)	Estimated distance second pass (m)	Error (m)	Accuracy Percentage (%)
1-2	2.733	2.523	0.210	92.32	2.517	0.216	92.10
3-5	1.253	1.167	0.086	93.14	1.124	0.129	89.70
4-6	0.690	0.580	0.110	84.06	0.595	0.095	86.23
5-7	0.684	0.640	0.044	93.57	0.620	0.064	90.64
6-7	4.345	3.962	0.383	91.19	3.930	0.415	90.45
2-8	0.912	0.800	0.112	87.72	0.815	0.097	89.36
4-8	2.449	2.258	0.191	92.20	2.256	0.193	92.12
3-9	1.845	1.529	0.316	82.87	1.593	0.252	86.34
8-9	0.864	0.862	0.002	99.77	0.764	0.100	88.43

TABLE 6.5: Actual values, measurements, and errors from the path displayed in Figure 6.14. The MAE is 16.2 cm for the first pass, and 17.3 cm for the second pass.

During the experiments, it was tested whether the algorithm could produce better results if more passes occurred during the point-capturing procedure. In the small room where the path formed a triangle, the accuracy is significantly increased during the second pass compared to the first. However, this was not the case in the experiment where the path formed a rectangle, indicating that even with additional passes the algorithm provided similar accuracy values. This was further confirmed in the large room experiment, where the accuracy was slightly lower in the second pass compared to the first pass. Additionally, experiments with rectangular paths were conducted to investigate whether the algorithm produced scaled points. Analysis of the results presented in Tables 6.2, 6.3, and 6.4 revealed no scaling errors in any of the tests performed.

6.5.5 Second Phase of Experiments

More experiments regarding the accuracy of the algorithm were conducted. In this phase, the areas that were observed were separated into small and large, and high- and low-featured. The high-featured area was on a tiled floor as shown in Figure 6.16, while the low-featured area was outside, with

the ground having little stones as shown in Figure 6.17. The small and large paths that were followed, in both high- and low-featured areas, are shown in Figure 6.18.

FIGURE 6.16: Texture of the high-featured area. They provide the algorithm with many features.

FIGURE 6.17: Texture of the high-featured area. They provide the algorithm with few features as it is a more uniform pattern.

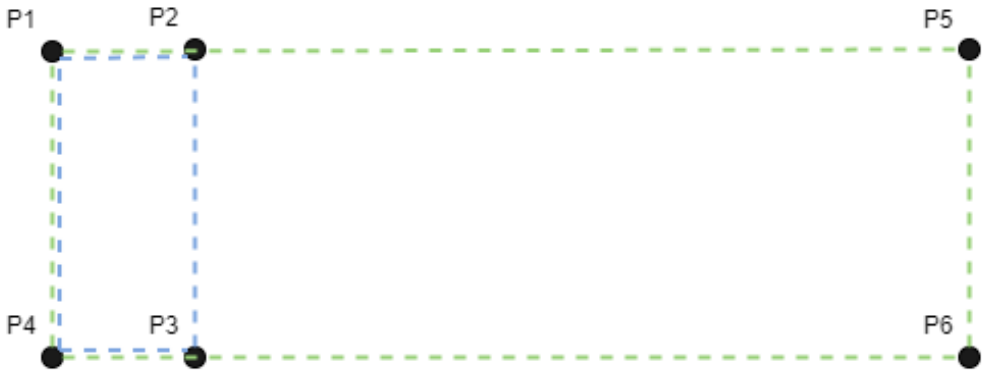


FIGURE 6.18: Paths followed and points captured during this phase of the experimentation. The path in blue represents the small area, while the path in green represents the large.

The purpose of this phase of experimentation is to examine in which environment the accuracy is higher. To achieve this, the point-capturing procedure was performed five times in each of the four areas. Every time, the path was followed one time for the loop closure to take place, and then the desired points were captured. For the small areas, these points are *P1*, *P2*, *P3*, and *P4*, while for the large areas, the *P1*, *P5*, *P6*, and *P4* points were captured.

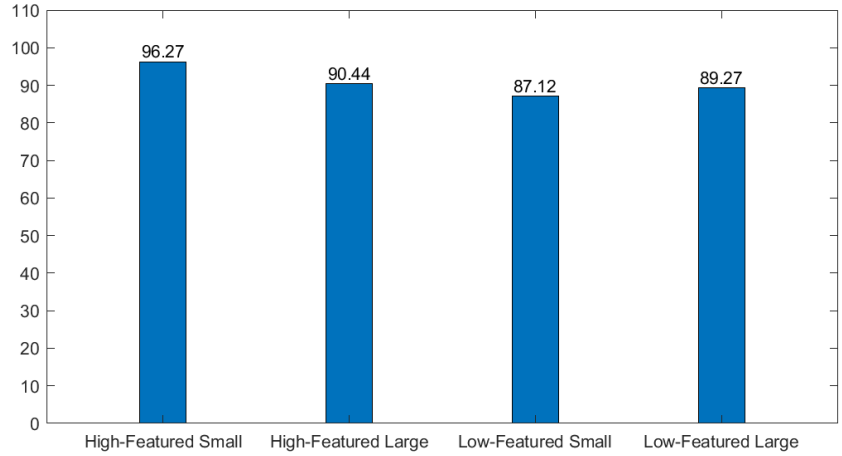


FIGURE 6.19: Accuracy of the system in the four areas.

It is concluded that the system produces better results in the area that contains more features and is small in size, as seen in Figure 6.19.

6.6 Chapter Summary

In conclusion, the chapter detailed the successful integration of system components, the challenges encountered during IMU integration, and the decision to exclude the IMU from the final system. The 3D printed case design was also described, highlighting the use of 3D printing for physical component integration. The experiments were conducted to verify the system's performance in different locations and environments that were selected due to specific characteristics. The created system performs with greater accuracy in high-featured and small-sized environments, in which the achieved accuracy was 96.27%, while in the other environments varied from 87.12% up to 90.44%. The system is more accurate in small areas because the error accumulation is fewer in small distances, and provides more accurate results in high-featured areas for the reason that there are more features to detect.

Chapter 7

Conclusions And Future Work

7.1 Conclusions

In conclusion, the main objectives were successfully accomplished, although with varying levels of accuracy, which can be attributed to the system being a first-generation system. The non-utilization of the IMU had a negative impact on the results obtained when compared to the accuracy reported in [1]. It was proven that the system performs optimally in relatively small high-featured areas, while its accuracy may decline in larger or low-featured areas. However, it was observed that in small, high-featured areas, the system can achieve a distance accuracy of 96.27%, even without utilizing the originally planned IMU. Moreover, the practical applications of these findings could be applied to create a better version of the system, and later be used in the fields of indoor positioning, among others.

7.2 Future Work

The results that occurred can lead to further improvements in future versions of the system, such as utilizing a more powerful embedded system, incorporating more accurate sensors, or experimenting with different algorithms. Another useful extension would be the addition of a GNSS receiver, that would result in extracting the coordinates from the captured points.

References

[1] Carlos Campos et al. “ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM”. In: *IEEE Transactions on Robotics* 37.6 (Dec. 2021), pp. 1874–1890. ISSN: 1941-0468. DOI: 10.1109/tro.2021.3075644. URL: https://doi.org/10.1109%2Ftro.2021.3075644.

[2] James Evans. “Dioptra”. In: *The History and Practice of Ancient Astronomy*. 1998. URL: https://books.google.gr/books?hl=el&lr=&id=nS51_7qbEWsC&oi=fnd&pg=PA3&dq=The+History+and+Practice+of+Ancient+Astronomy&ots=c5jkSgnzpe&sig=dnF9H33UfY2DzRbpQn5pGCV8GJc&redir_esc=y#v=onepage&q=The%20History%20and%20Practice%20of%20Ancient%20Astronomy.

[3] M. J. T. Lewis. “The groma”. In: *Surveying Instruments of Greece and Rome*. Cambridge University Press, 2001, 120–133. DOI: 10.1017/CBO9780511483035.008. URL: https://www.cambridge.org/core/books/abs/surveying-instruments-of-greece-and-rome/groma/245DE0E97E86681142200AFE4D0288A0.

[9] Gulnur Selda Kuruoglu, Melike Erol, and Sema Oktug. “Three Dimensional Localization in Wireless Sensor Networks Using the Adapted Multi-Lateration Technique Considering Range Measurement Errors”. In: *2009 IEEE Globecom Workshops*. 2009, pp. 1–5. DOI: 10.1109/GLOCOMW.2009.5360717. URL: https://ieeexplore.ieee.org/document/5360717.

[10] Alan Bensky. “Chapter 14 - Technologies and applications”. In: *Short-range Wireless Communication (Third Edition)*. Ed. by Alan Bensky. Third Edition. Newnes, 2019, pp. 387–430. ISBN: 978-0-12-815405-2. DOI: https://doi.org/10.1016/B978-0-12-815405-2.00014-2. URL: https://www.sciencedirect.com/science/article/pii/B9780128154052000142.

[11] Wen Liu et al. “Survey on CSI-based Indoor Positioning Systems and Recent Advances”. In: *2019 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. 2019, pp. 1–8. DOI: 10.1109/IPIN.2019.8911774. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7487628&isnumber=7487087.

[12] Lin Qi et al. “Current Status and Future Trends of Meter-Level Indoor Positioning Technology: A Review”. In: *Remote Sensing* 16.2 (2024). ISSN: 2072-4292. DOI: 10.3390/rs16020398. URL: https://www.mdpi.com/2072-4292/16/2/398.

[13] Xin Zhou et al. “Indoor positioning with multi-beam CSI of commercial 5G signals”. In: *Urban Informatics* 3 (2024), p. 1. ISSN: 2731-6963. DOI: 10.1007/s44212-023-00034-4. URL: https://doi.org/10.1007/s44212-023-00034-4.

[14] Yuan Zhuang et al. “Bluetooth Localization Technology: Principles, Applications, and Future Trends”. In: *IEEE Internet of Things Journal* 9.23 (2022), pp. 23506–23524. ISSN: 2327-4662. DOI: 10.1109/JIOT.2022.3203414. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9878154&isnumber=9955314.

[15] Mahmoud Elsanhoury et al. “Precision Positioning for Smart Logistics Using Ultra-Wideband Technology-Based Indoor Navigation: A Review”. In: *IEEE Access* 10 (2022), pp. 44413–44445. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3169267. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9761257&isnumber=9668973.

[16] “IEEE Standard for Information Technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications”. In: *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)* (2016), pp. 1–3534. DOI: 10.1109/IEEESTD.2016.7786995. URL: https://ieeexplore.ieee.org/document/7786995.

[17] Veronika Hromadova, Peter Brida, and Juraj Machaj. “Design of Acoustic Signal for Positioning of Smart Devices”. In: *Sensors* 23.18 (2023). ISSN: 1424-8220. DOI: 10.3390/s23187852. URL: https://www.mdpi.com/1424-8220/23/18/7852.

[18] Tong Liu et al. “Pseudolites to Support Location Services in Smart Cities: Review and Prospects”. In: *Smart Cities* 6.4 (2023), pp. 2081–2105. ISSN: 2624-6511. DOI: 10.3390/smartcities6040096. URL: https://www.mdpi.com/2624-6511/6/4/96.

[19] Huy Q. Tran and Cheolkeun Ha. “Machine learning in indoor visible light positioning systems: A review”. In: *Neurocomputing* 491 (2022), pp. 117–131. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2021.10.123. URL: https://www.sciencedirect.com/science/article/pii/S092523122200306X.

[20] Huy Quang Tran and Cheolkeun Ha. "High Precision Weighted Optimum K-Nearest Neighbors Algorithm for Indoor Visible Light Positioning Applications". In: *IEEE Access* 8 (2020), pp. 114597–114607. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3003977. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9121990&isnumber=8948470.

[21] Valter Pasku et al. "Magnetic Field-Based Positioning Systems". In: *IEEE Communications Surveys & Tutorials* 19.3 (2017), pp. 2003–2017. ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2684087. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7880609&isnumber=8013866.

[29] Tully Foote. "tf: The transform library". In: *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*. Open-Source Software workshop. Apr. 2013, pp. 1–6. DOI: 10.1109/TePRA.2013.6556373. URL: https://wiki.ros.org/Papers/TePRA2013_Foote.

[30] D.G. Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. Sept. 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=790410&isnumber=17141.

[31] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "SURF: Speeded Up Robust Features". In: *Computer Vision – ECCV 2006*. Ed. by Ales Leonardis, Horst Bischof, and Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417. ISBN: 978-3-540-33833-8. DOI: 10.1007/11744023_32. URL: https://rdcu.be/dpcTW.

[32] Miroslav Trajković and Mark Hedley. "Fast corner detection". In: *Image and Vision Computing* 16.2 (1998), pp. 75–87. ISSN: 0262-8856. DOI: https://doi.org/10.1016/S0262-8856(97)00056-5. URL: https://www.sciencedirect.com/science/article/pii/S0262885697000565.

[33] Michael Calonder et al. "BRIEF: Binary Robust Independent Elementary Features". In: *Computer Vision – ECCV 2010*. Ed. by Kostas Daniilidis, Petros Maragos, and Nikos Paragios. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 778–792. ISBN: 978-3-642-15561-1. DOI: 10.1007/978-3-642-15561-1_56. URL: https://doi.org/10.1007/978-3-642-15561-1_56.

[34] Ethan Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. Institute of Electrical

and Electronics Engineers (IEEE), 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544. URL: http://ieeexplore.ieee.org/document/6126544/.

[35] Dinar Sharafutdinov et al. "Comparison of modern open-source Visual SLAM approaches". In: *Journal of Intelligent and Robotic Systems* 107 (3 Mar. 2023), p. 43. DOI: 10.1007/s10846-023-01812-7. URL: https://doi.org/10.1007/s10846-023-01812-7.

[36] Thien-Minh Nguyen et al. "VIRAL SLAM: Tightly Coupled Camera-IMU-UWB-Lidar SLAM". In: *ArXiv* abs/2105.03296 (Oct. 2021). URL: https://doi.org/10.48550/arXiv.2105.03296.

[37] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. "ORB-SLAM: A Versatile and Accurate Monocular SLAM System". In: *IEEE Transactions on Robotics* 31.5 (Oct. 2015), pp. 1147–1163. ISSN: 1941-0468. DOI: 10.1109/TRO.2015.2463671. URL: https://ieeexplore.ieee.org/document/7219438.

[38] Raúl Mur-Artal and Juan D. Tardós. "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras". In: *IEEE Transactions on Robotics* 33.5 (Oct. 2017), pp. 1255–1262. ISSN: 1941-0468. DOI: 10.1109/TRO.2017.2705103. URL: https://ieeexplore.ieee.org/document/7946260.

[39] Dinar Sharafutdinov et al. "Comparison of modern open-source Visual SLAM approaches". In: *Journal of Intelligent & Robotic Systems* 107.3 (2023), p. 43. ISSN: 1573-0409. DOI: 10.1007/s10846-023-01812-7. URL: https://doi.org/10.1007/s10846-023-01812-7.

[40] Tong Qin, Peiliang Li, and Shaojie Shen. "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator". In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 1004–1020. ISSN: 1941-0468. DOI: 10.1109/TRO.2018.2853729. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8421746&isnumber=8437110.

[45] Oliver J. Woodman. *An introduction to inertial navigation*. Tech. rep. UCAM-CL-TR-696. 15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom: Computer Laboratory, University of Cambridge, Aug. 2007, p. 37. URL: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-696.pdf.

[46] Nikolas Trawny and Stergios I. Roumeliotis. *Indirect Kalman Filter for 3D Attitude Estimation*. Tech. rep. TR-2005-002. 200 Union St. S.E., Minneapolis, MN 55455: Dept. of Computer Science and Engineering, University of Minnesota, Mar. 2005, p. 25. URL: http://mars.cs.umn.edu/tr/reports/Trawny05b.pdf.

[47] Leslie Barreda Pupo. "Characterization of Errors and Noises in MEMS Inertial Sensors Using Allan Variance Method". MA thesis. Universitat Oberta de Catalunya, 2016.

[48] Joern Rehder et al. "Extending kalibr: Calibrating the extrinsics of multiple IMUs and of individual axes". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. May 2016, pp. 4304–4311. DOI: 10.1109/ICRA.2016.7487628. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7487628&isnumber=7487087.

[50] Paul Furgale, Joern Rehder, and Roland Siegwart. "Unified temporal and spatial calibration for multi-sensor systems". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Nov. 2013, pp. 1280–1286. DOI: 10.1109/IROS.2013.6696514. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6696514&isnumber=6696319.

[51] Jérôme Maye, Paul Furgale, and Roland Siegwart. "Self-supervised calibration for robotic systems". In: *2013 IEEE Intelligent Vehicles Symposium (IV)*. June 2013, pp. 473–480. DOI: 10.1109/IVS.2013.6629513. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6629513&isnumber=6629437.

[52] Luc Oth et al. "Rolling Shutter Camera Calibration". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2013. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2013/html/Oth_Rolling_Shutter_Camera_2013_CVPR_paper.html.

[53] S. Garrido-Jurado et al. "Automatic generation and detection of highly reliable fiducial markers under occlusion". In: 47.6 (2014), pp. 2280–2292. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2014.01.005. URL: https://www.sciencedirect.com/science/article/pii/S0031320314000235.

External Links

[4] *Inclinometer*. URL: https://en.wikipedia.org/wiki/Inclinometer.

[5] *Theodolite*. URL: https://en.wikipedia.org/wiki/Theodolite.

[6] *Leica Shop Catalog*. URL: https://shop.leica-geosystems.com/.

[7] *Leica BLK2GO link*. URL: https://shop.leica-geosystems.com/leica-blk/blk2go/buy.

[8] *Leica BLK360 user manual*. URL: https://shop.leica-geosystems.com/sites/default/files/2022-01/853811_Leica_BLK360_UM_v4.0.0_en.pdf.

[22] *Pixhawk autopilot*. URL: https://docs.px4.io/main/en/flight_controller/pixhawk_series.html.

[23] *Realsense T265 stereo camera with IMU*. URL: https://www.intelrealsense.com/wp-content/uploads/2019/09/Intel_RealSense_Tracking_Camera_Datasheet_Rev004_release.pdf.

[24] *Realsense D435 depth stereo camera*. URL: https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/Intel-RealSense-D400-Series-Datasheet.pdf.

[25] *Dell Inspiron 15 7590 documentation*. URL: https://dl.dell.com/topicspdf/inspiron-15-7590-laptop_users-guide_el.pdf.

[26] *Raspberry Pi 4 documentation*. URL: https://www.raspberrypi.com/documentation/computers/getting-started.html.

[27] *Raspberry Pi 4 specifications*. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.

[28] *ROS*. URL: https://www.ros.org/.

[41] *wxPython Library*. URL: https://wxpython.org/index.html.

[42] *Camera-IMU calibration tutorial from Kalibr*. URL: https://github.com/ethz-asl/kalibr/wiki/camera-imu-calibration.

[43] *Camera-IMU calibration tutorial from ORB-SLAM3*. URL: https://raw.githubusercontent.com/UZ-SLAMLab/ORB_SLAM3/master/Calibration_Tutorial.pdf.

[44] *Github page of Kalibr*. URL: https://github.com/ethz-asl/kalibr.

[49] *Github page of Allan Variance ROS*. URL: https://github.com/ori-drs/allan_variance_ros.

[54] *ROS convention for axes orientation*. URL: https://www.ros.org/reps/rep-0103.html.

[55] *ROS convention for transform frames*. URL: https://www.ros.org/reps/rep-0105.html.

[56] *Sensor_msgs/Imu*. URL: https://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/Imu.html.

[57] *Geometry_msgs/TransformTransformStamped*. URL: https://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/TransformStamped.html.

[58] *Geometry_msgs/Transform*. URL: https://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Transform.html.

[59] *Issue #736 from the GitHub page of ORB-SLAM3*. URL: https://github.com/UZ-SLAMLab/ORB_SLAM3/issues/736.